IMPLEMENTATION OF LPC BASED SPEECH PROCESSING ALGORITHMS ON ADSP 2100

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF TECHNOLOGY

by

PEEYUSH PANDEY

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

September, 1992

CERTIFICATE

It is certified that the work contained in the thesis entitled "IMPLEMENTATION OF LPC BASED SPEECH PROCESSING ALGORITHMS ON ADSP 2100", has been carried out by Peeyush Pandey under my supervision and that this work has not been submitted elsewhere for a degree.

(M.U.Siddiqi) 21.9.92

Professor

Department of Electrical Engineering

I.I.T. KANPUR.

September, 1992.

EE-1992-M-PAN-IMP

114542

ACKNOWLEDGEMENT

I am grateful to my thesis supervisor Prof M.U.Siddiqi for suggesting me this field of study and his overall guidance throughout the work. His stress on the quality of work done will always be an inspiration.

I would like thank Ramprasad and Hariprasad for the help they had given me during my thesis work. Arun, Udaya, Madhu and Venkatesh not only helped me but were more like elder brothers to me.

Mahajan, Balvinder, Galgali, Subbarao, Chanchal and Deepak Gupta were ever ready to help me and without their gracious help I do not know how I would have completed this thesis.

IIT was made beautiful by Alok, C–top, Mohalik, Samudra, Anurag, and Hemant. I thank Nandini for her company during the last phase of my work. I have learnt a lot from all of them.

Dedicated to

YOU

ABSTRACT

This thesis aims at implementing some of the speech processing algorithms, which can be used as central building blocks for the development of a speech processing system based around the ADSP 2100 DSP microprocessor. Code excited linear prediction (CELP), pole–zero analysis, and formant based analysis with pattern matching of voice on the frequency axis are the algorithms that have been implemented. The CELP algorithm forms the main unit of a coder–decoder. The pole–zero analysis forms the central unit for a speech recognition system and the pattern matching algorithm forms the basic unit for a speaker verification/identification system. The program development, testing, and debugging have been done on the ADSP 2100 simulator and the algorithms have been implemented on the ADSP 2100 evaluation board and comparisons have been made with the implementation in a high level language "C". Suitability of the implementations for real time cases has been discussed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 MOTIVATION.

Man has a unique method of communication and that is speech. But human speech is very restricted, in the sense that only human beings who know that particular language understand it, and it can be heard only if the listener is nearby.

It has been always been a major interest, for scientists to overcome these limitations. Then there are other requirements which draw attention, for example speech recognition, and speaker recognition. For all these purposes processing of speech is required. With the advantages of complexity and speed on its side, digital speech processing now is the most sought after area in speech processing research. There are six major areas of digital speech processing :

- Transmission and storage : Telephone is a very well known method of speech transmission but it is desirable to decrease the bandwidth required to transmit the speech signal. Furthermore, a need has arisen for systems which digitize speech at as low a bit rate as possible. Also, there is the possibility of extremely sophisticated encryption of the speech signal which favour the use of digital processing.

- Speech synthesis systems : Much of the interest in this area is motivated by the need for economical digital storage of speech for computer voice response systems.

- Speaker verification and identification systems : The techniques here involve authentication or identification of a speaker from a large ensemble of possible speakers. It has its many uses especially in restricted areas and forensic applications.

• Speech recognition system : Here the technique is to basically understand the uttered words. For example, a machine able to understand the speech input instructions. The major problem here is to understand same words uttered by different speakers.

• Aids for the handicapped : The application here concerns processing of speech signal to make the information available in a form which is better matched to a handicapped person's abilities than is normally available.

• Enhancement of speech quality : Processing of speech signals to improve the quality of speech that is available. For example some part of speech is missing, it can be reconstructed or the filtering of noisy speech to make it clear.

1.2 SCOPE OF THE WORK

We now know of the present areas of speech processing applications. All the applications drew our interest. Our aim was to implement some of the algorithms in these applications areas and hence have a speech processing system. To implement the algorithms we chose the ADSP 2100 digital signal microprocessor. Why we chose this particular microprocessor? The reasons are not hard to describe.

The ADSP 2100 is a single chip microprocessor optimized for DSP and other high speed numeric processing applications. It integrates computational units, data address generators and program sequencer in a single device. ADSP 2100 makes efficient use of external memories for program and data storage, freeing silicon area for increased processor performance. It combines the performance of a bit slice/building block system with the ease of design and development of a general purpose microprocessor.

ADSP 2100 operates at 8.192MHz and every instruction executes in a single 125ns cycle. Fabricated in a high speed 1.5 micron double layer metal CMOS process, ADSP 2100 dissipates less than 600 mW.

ADSP 2100's flexible architecture and comprehensive instruction set support a high degree of operational parallelism and in one cycle it can:

- generate the next program address.
- fetch the next instruction.
- perform one or two data moves.
- update one or two data address pointers.
- perform a computational operation.

We started with digital transmission of speech. Therefore, our first aim was to implement a coder decoder (CODEC) which could code speech for transmission and decode it at the receive end. Through the implementation of this we would achieve the first applications listed i.e. transmission. We successfully implemented the algorithm for the coding and subsequent decoding of speech. We used code excited linear prediction (CELP) of speech to implement our CODEC.

We then wanted to implement a speech recognition algorithm. As our CODEC is based on the linear prediction (LP) technique we thought of implementing our recognition system based on LP technique as well. Pole–zero classification of speech is a very good way of speech recognition. We therefore chose an algorithm that finds the pole and zeros from speech using linear predictive coding (LPC) method. For a given sample of speech using LPC the algorithm finds the poles and from them the zeros. Using these, the poles and zeros can be generated, stored and classified to make a speech recognition system. We have implemented the algorithm and it generates the poles and zeros from the given speech samples.

Since we had a speech coding decoding system and a speech recognition system we wanted to implement something to do with speaker verification and identification. Pattern matching of voice is a well known method for speaker verification and identification. It occupies the central theme in both the uses of identification and

verification. Pattern matching of voice on the time axis is the most well known method for this purpose. We felt that pattern matching on the frequency axis should be attempted. Not because it is something new but because it has some advantages over the time axis pattern matching methods. This algorithm has also been implemented. It calculates the distance measure between two speech patterns. This distance measure can be used for the purpose of speaker verification or identification.

1.3 IMPLEMENTATIONS

In our implementations we have a major part of the above described speech processing applications. We have the central system for three major speech processing applications. They are:

- Speech coding and decoding : A CODEC implementation based on code excited linear prediction (CELP) of speech.
- Speech Recognition : A program for the pole–zero calculation of speech. This program can be first used to generate poles and zeros for speech and pole zero tables can be prepared. Later on speech input can be used to generate poles and zeros, and based on classification a speech recognition system fully implemented.
- Speaker verification and verification : A program for the formant based pattern matching of speech. Here the program is used to generate the control parameters and store them. Speech inputs later can be compared using these tables for identification and verification.

1.4 ORGANIZATION OF THE THESIS.

Chapter 2 discusses the complete details of the ADSP 2100 microprocessor and its associated systems for the development of DSP based application systems. The chapter talks in detail the architecture of the ADSP 2100 DSP microprocessor. The use of

the system builder, assembler and the linker to develop the final executable program is explained. It also discusses the working and use of the evaluation board and the simulator for the ADSP 2100.

Chapter 3 is a review chapter on the narrow band speech processing techniques. In the chapter the basic narrow band model of the speech production process is dealt with. This chapter also discusses the practical and experimental contemporary voice bandwidth compression techniques.

Chapter 4 discusses CELP in detail. First the algorithm and then various methods of speeding up the algorithm have been taken up. It also discusses algorithm implementation on the ADSP 2100.

Chapter 5 discusses speech recognition and how pole–zero analysis can be used for this purpose. The particular algorithm used to calculate poles and zeros has been described. Finally its implementation on the ADSP 2100 is explained.

Chapter 6 discusses speaker recognition and identification, and how pattern matching achieves the purpose. The chapter describes the pattern matching algorithm and its implementation on the ADSP 2100.

Chapter 7 discusses the final results, compares the performance with the performance on a PC. It analyses the real time aspects of the problems dealt with and concludes it with the discussion on scope for future work.

CHAPTER TWO

ADSP 2100 DIGITAL SIGNAL MICROPROCESSOR

2.1 INTRODUCTION

The development of Very Large Scale Integration (VLSI) in the field of digital electronic circuitry has brought about a major change in all the analog based processes. All the fields which required fast processing turn to VLSI for help. Signal processing which requires a lot of processing at fast speed is not untouched. The development of both the fields along side has brought about the development of VLSI chips specific to DSP applications. There are a large number of such DSP chips at present available in the market and the ADSP 2100 from Analog Devices is one such DSP specific chip.

Our speech processing system is based on this particular DSP microprocessor and therefore we first of all discuss the microprocessor itself.

2.2 ADSP 2100 ARCHITECTURE OVERVIEW [1]

Figure 2.1 shows the internal architecture of ADSP 2100 DSP microprocessor. ADSP 2100 microprocessor has five internal busses to handle data communication within the microprocessor. The five busses are *Program Memory Address* (*PMA*) bus, *Program Memory Data* (*PMD*) bus, *Data Memory Address* (*DMA*) bus, *Data Memory Data* (*DMD*) bus, and *Result* (*R*) bus. The program and.data memory busses (i.e. PMA, PMD, DMA and DMD) extend out of the chip to provide connections and data communication with external memories. The Result (R) bus is linked with all the computational units within the processor and provides a data link between them. One is

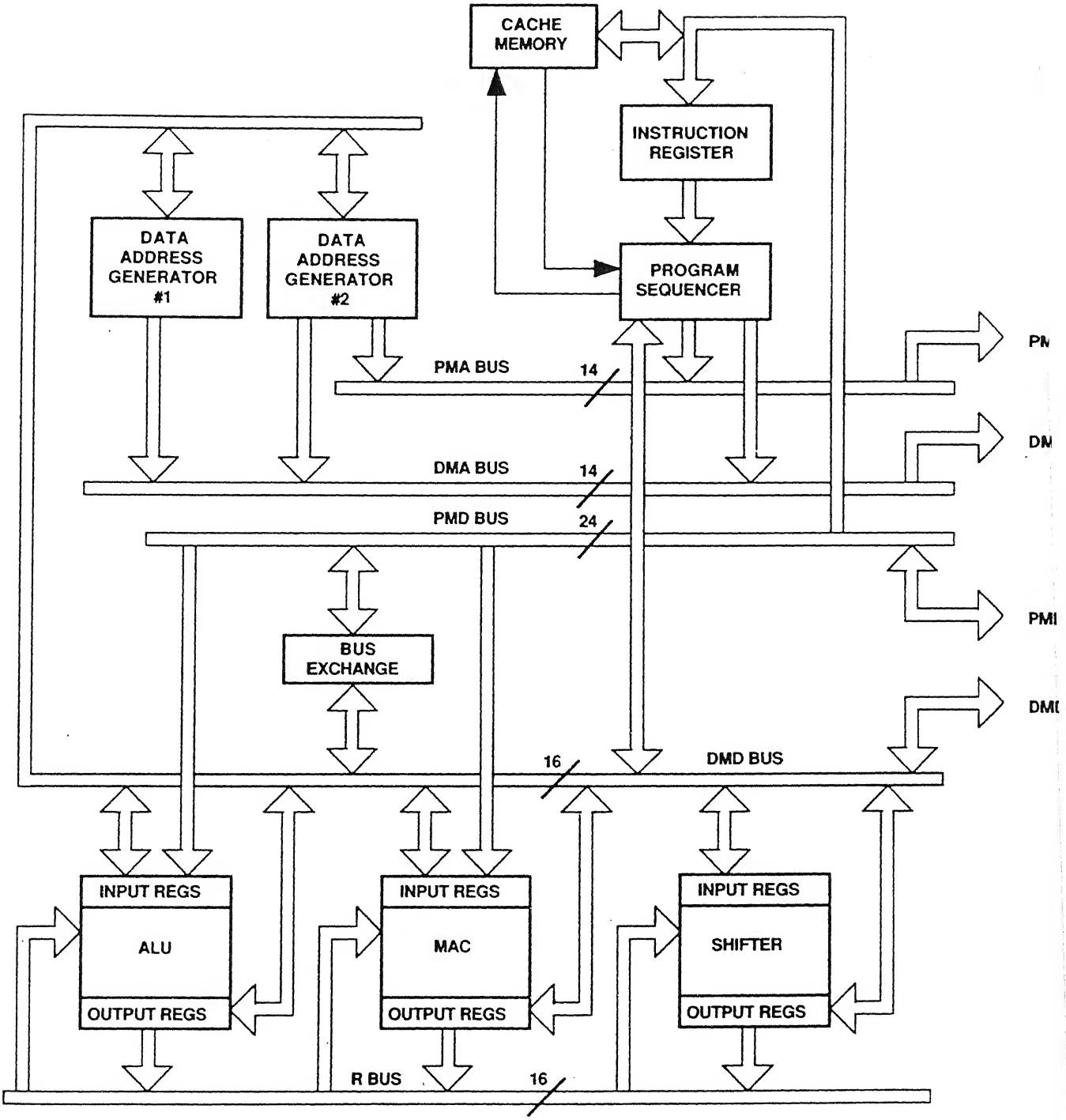


Figure 2.1 Internal Architecture of ADSP 2100

tempted to mention at this stage the advantage of configuring two data memories as done in this architecture. As DSP applications are highly data intensive, a lot of memory access has to be done so a major saving of time will be achieved if this memory access time is reduced. The main advantage with two data memories therefore is that the microprocessor can access both the memories in a single fetch cycle. This therefore saves a lot of memory access time.

The processor supports two external memories, *Program Memory* (PM) and *Data Memory* (DM) as per the memory bus configuration discussed above. The program memory stores 24 bit ADSP 2100 instructions (i.e.CODE). Program Memory Data bus hence is 24 bits wide. Inside the processor the Program Memory Data bus has two paths, one leading to the instruction decode section and the other to the computational section. This is done so that the program memory can then serve a dual purpose and store data as well. There is a Program Memory Data Access (PMDA) signal line in the processor. This signal is asserted when data is being fetched from the program memory. Program memory address bus is 14 bits wide. The processor uses the PMDA line as a 15th address line and so the microprocessor can access 16K program memory instruction code and 16K program memory data (let us mention here that the registers in the processor are 16 bits wide and hence the Program Memory Data area in ADSP 2100 based systems are usually configured 16 bits wide though the program memory data bus is 24 bits wide as mentioned above). All the memory locations in the microprocessor are both readable and writeable and can be either ROM or RAM.

The Data Memory Address bus (DMA) is 14 bits wide so, a direct access of 16K words is possible. The data path is 16 bits wide as the registers are 16 bits wide. Data Memory Address bus (DMA) is used to access data, both from memory and peripheral devices (in microprocessor terminology this type of access from peripheral it is termed as a memory mapped I/O). Asynchronous operation of the chip is possible using the Data

Memory Acknowledge (DMACK) signal. This allows the processor to communicate with slower peripheral devices (the processor inserts wait state cycles until the DMACK signal is asserted by the peripheral i.e.the device acknowledges the task to be over).

Internally the processor can be divided into three major sections, *program sequencer section, data address generation section* and *computation section.* The function of the program sequencer is to generate instruction addresses, thus controlling program flow of the processor. In the program sequencer section there is an internal cache, which is 16 instructions wide and it maintains a history of previously executed instructions. This therefore becomes an alternate source of instruction code when executing a loop which is wholly contained in the cache. This is useful as cache is always faster than other external memory devices. The cache operation is transparent to the user. There are two address generators (DAG1 and DAG2) in the address generator section. They handle indexed addressing with post modify. Each address generator keeps track of four pointers. After the pointer is used to access the external data, it is modified by a specified value. The value for updating is in one of the 'M' registers. A length value associated with each pointer implements arithmetic modulo addressing for circular buffers. The value for this is kept in one of the 'L' registers. The two generators are identical except that DAG1 has a bit reversal option at output (this is useful in FFT based applications) and only DAG2 can drive PMA lines (both generators can drive DMA lines). The third unit is the computational unit. The computational unit can be further subdivided into three units. *The Arithmetic Logic unit* (ALU), *the Multiplier Accumulator unit* (MAC) and *the Shifter.* All computational units can operate on 16 bit data directly. Each unit functions independently of the others but only one unit operates in a cycle. The ADSP 2100A can accept up to four asynchronous interrupts. These interrupts are internally prioritized and maskable.

The list of registers with classification and their functions and the data size

they hold, is given in Appendix A. The registers are mainly classified as *Data registers*, *Index registers*, *Modify registers*, *Shifter registers*. Basically the names of registers themselves will give a hint as to what may be the function of the specified registers.

We now have a overview of the architecture of the ADSP 2100. We will not be listing all the possible assembly language instructions here, they can be given in the ADSP 2100 software manual. A list of the types of instructions of the ADSP 2100 is given in Appendix B.

2.3 DEVELOPMENT FLOW, ASSEMBLER, SYSTEM BUILDER and LINKER [1]

Figure 2.2 shows the flow chart of the development cycle. First of all we begin by defining the target hardware environment. The target hardware environment means how the actual hardware system is going to be configured. The file specifying the system (i.e the hardware environment) is System Specification file (.SYS). It is written using system builder directives. The system builder reads this file and generates one output file, the Architecture Description file (.ACH). This file passes on information about the target hardware to the Linker and Simulator.

We then begin code generation by using assembly source code modules (the files for these modules are Source Code files (.DSP) files). An assembly module is a file in the ADSP 2100 assembly language and is made by the user if he wants to modularise one's programs. It can be a calling program, subroutine, data buffer declaration section, or any combination of these. Each assembly source code module is assembled separately by the ADSP 2100 Assembler. The Assembler translates these source code modules into object code modules. An assembly source code module is created using ADSP 2100 assembly language and defining variables, data buffers and symbolic constants using assembler directives. Separately assembled modules are later linked together to form a running system. The Assembler reads a Source Code file (.DSP) and generates four output files

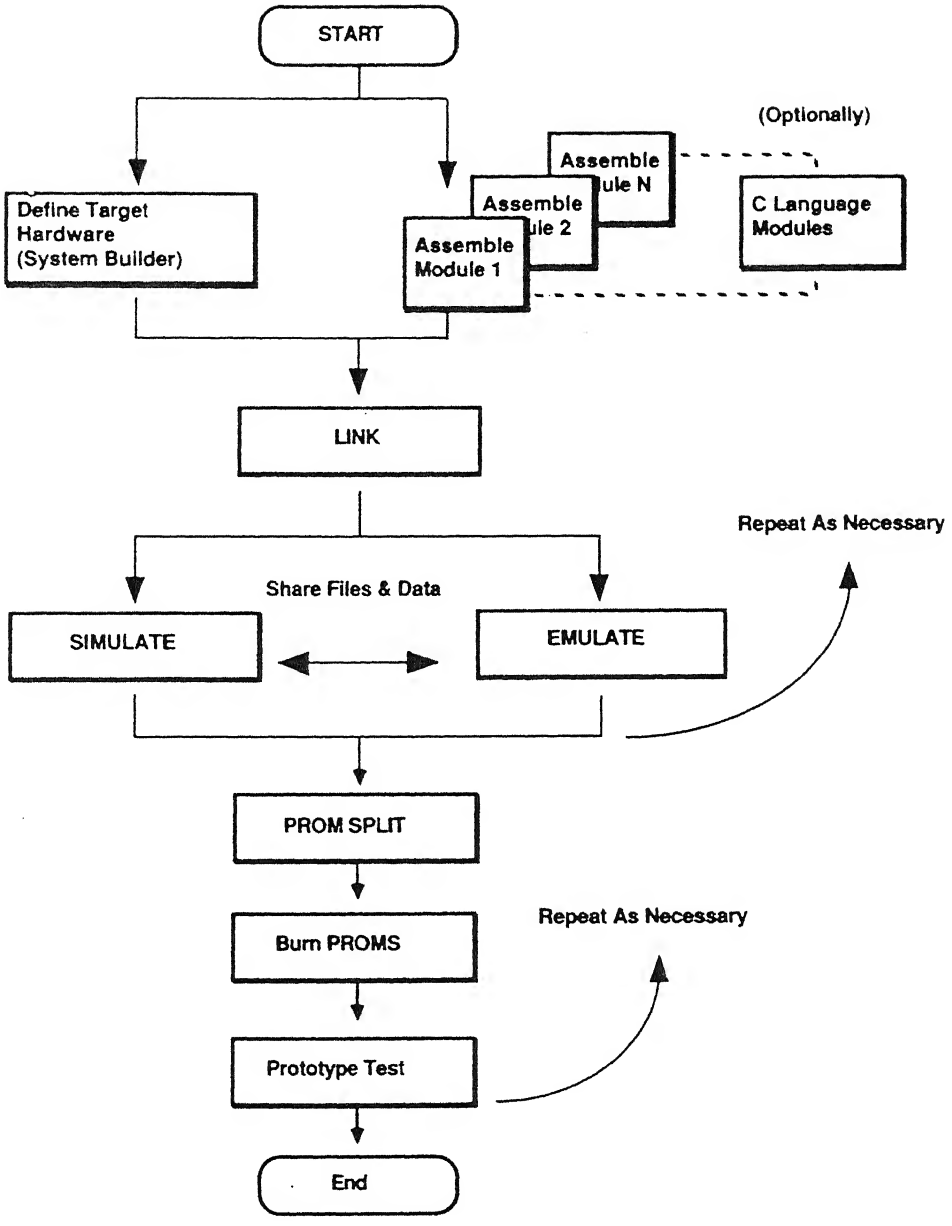


Figure 2.2 Development Flow

with the same root name. The files generated are the Object file (.OBJ), a Code file (.CDE), an Initializing file (.INT), and a List file (.LST). The Object file, Code file and Initialization files are passed on to the Linker. The Object file contains information regarding memory allocation and symbol declarations in the program. The Code file contains instruction opcodes with the unresolved symbols marked. The Initialization file contains initialization information for data buffers. The List file which is optional is for documentation and message passing.

Using assembler directives in the source code file, one can include other source code files and inform the Linker of initialization data files in the assembly process. These files are read by the Assembler and processed together with the original source file. We thus see that the assembler supports macro capability.

The ADSP 2100 Linker generates a complete executable program by linking together program modules which were assembled separately. The output of the Linker is used by the Emulator, Evaluation Board, Simulator and PROM Splitter. Together with the Assembler output files, Initialization data files and Architecture description files the Assembler creates one complete executable code file by resolving external references and assigning addresses to relocatable code and data spaces. Of the output files of the Linker the Memory image file (.EXE) contains the actual program and data memory images after the linkages. The optional Map listing file (.MAP) outputted by the linker assists the designer in interpreting results of the linkage. The third file output by the Linker is Symbol Table file (.SYM) which lists all the symbols encountered by the Linker, their absolute values and their scope of reference.

2.3.1 Use of System Builder, Assembler and Linker

We would like to know how the System Builder, Assembler and Linker are used to generate the files described above.

System Builder

The System Builder is invoked from the host using the command:

DSPPB filename.ext<cr>

where filename.ext is the System Specification File created by us. By default the filename extension is taken as .SYS. In the System Specification file, symbolic names are assigned to the following entities: the system description, I/O ports, and the memory segments. All symbolic names must be unique. A set of keywords reserved for system description cannot be used as symbolic names.

We will not go into the detail of describing each system builder directive as it can be found in the software manual of the ADSP 2100. For the list of the System Builder directives with a short explanation see Appendix C.

Assembler

The Assembler is invoked, from the host system, and its command form is:

DSPPA filename.ext [switches] <cr>

where filename.ext is the source code file. The filename extension is optional and defaults to .DSP. Other data and source code files can be included in the assembly process by using certain assembly directives. The assembly directives are listed in Appendix D. The switches help perform certain specialized functions.

Linker

The ADSP 2100 Linker generates a complete executable program by linking together program modules which were assembled separately.

The linker is invoked in the host system using the command:

DSPPL –i file_all [–switch..]

or DSPPL file1 [file2] [file3]..[–switch]

in the second form which we preferred one has to explicitly name the files to be linked.

Figure 2.3 shows the whole process from the start when one has developed the initial files (i.e. the .DSP and .SYS files) to the final stage of output files from the linker. The Memory image (.EXE) file contains actual program and data memory images after the linkages. The debug Symbol Table file (.SYM) contains a list of all the symbols encountered by Linker, their absolute values and their scope of reference.

2.4 THE SIMULATOR AND EVALUATION BOARD [1,2]

Before going into development of programs let us discuss some of the features of the systems on which we will be developing these programs.

2.4.1 The Simulator

The ADSP 2100 Simulator helps one to verify their design applications based on the ADSP 2100 before any hardware development is taken up. To do the task of simulation, the Simulator first, reads the Architecture Description file (.ACH), Memory Image file (.EXE), and Symbol Table file (.SYM). All these files are created first using the Assembler, Builder and Linker (as described above). The Architecture Description file is input to the Simulator to start an ADSP 2100 simulation session by configuring the simulated system as per target architecture. The Memory Image file is loaded when the simulator gives a prompt to do so. The Symbol Table file is loaded automatically along with the Memory Image file.

The Simulator is highly interactive and screen oriented. Using the Symbol Table file, the Simulator is able to understand symbols given by the user i.e it interacts symbolically. One can make references to variables and program labels using symbols defined in the source code instead of actual numeric addresses. The Simulator operates in three modes:

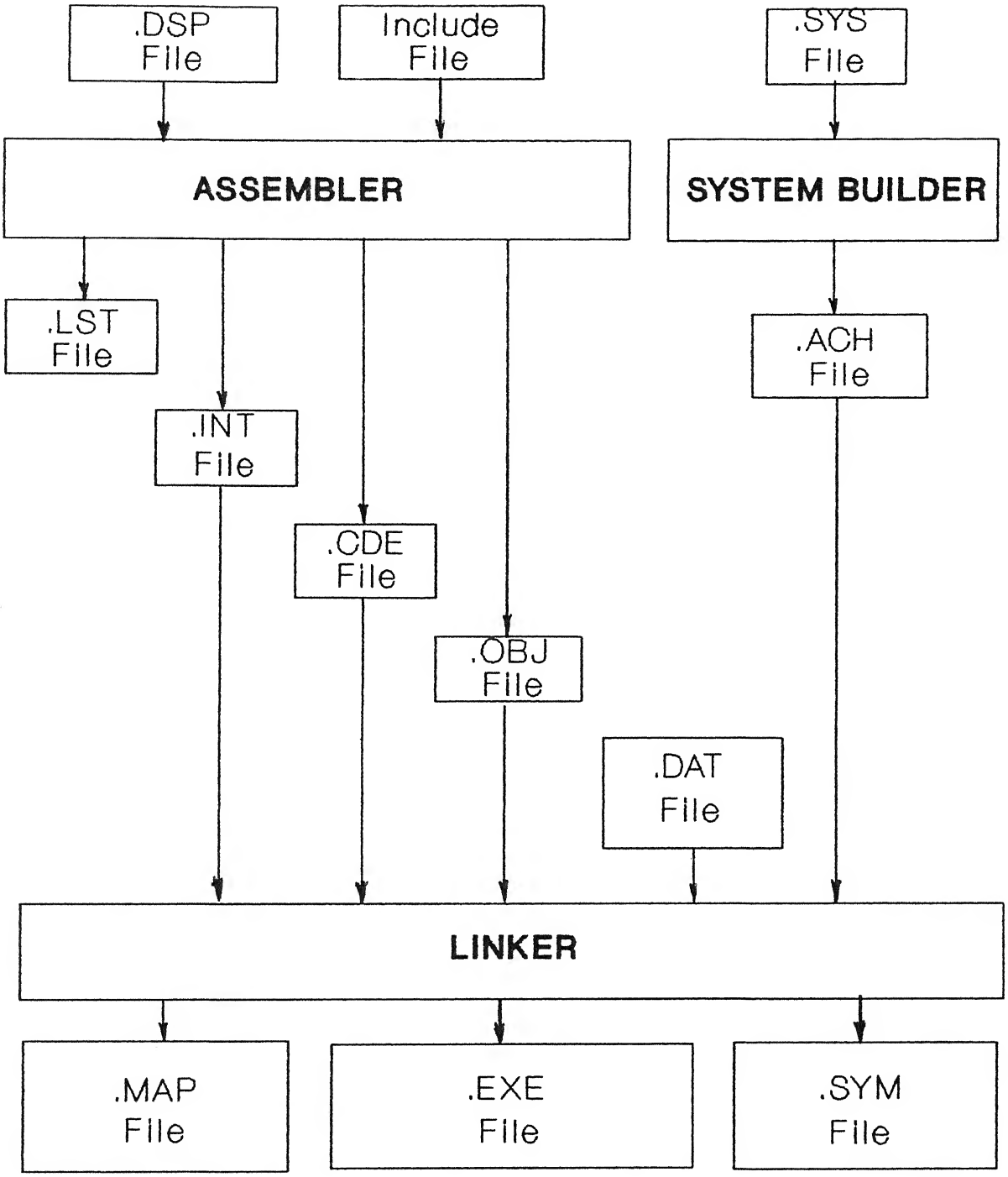


Figure 2.3 Developing Executable Programs From the .DSP & .SYS files

• Emulator Mode : Simulator runs at full speed and halts only when it encounters a break command (it could be the end of the program i.e a user installed trap) or a key pressed from the key board. While the Simulator is running in this mode, the display is updated every 256 cycles and when it halts the full screen is updated.

• Extend Mode : In this mode, all the display contents are updated after each instruction is executed, and instructions are disassembled on the screen as they are executed.

• Single–Step Mode : In this mode the RUN command executes one instruction at a time and halts at the end of each instruction. All displays are updated and instructions disassembled as they execute.

The Simulator offers six display modes. These display modes are:

• Register display : displays all the registers.

• Data memory display : displays the data memory contents

• Program memory display : displays contents of program memory

• Cache memory display : displays disassembled contents of the cache memory.

• Stack display : display shows the contents of the count stack, loop stack, condition codes and the PC (program counter) stack

• Trace buffer display : The Simulator keeps a running history of past external bus states in a 4K trace buffer. Any part of the trace buffer can be looked into by specifying the time offset from the last execution.

A list of simulator commands is given in Appendix E. The simulator commands are self explanatory, in the sense that one will understand what are the possible operations that can be performed on the programs that are being tested.

2.4.2 The Evaluation Board

The ADSP 2100 Evaluation Board is a stand alone system consisting of an ADSP 2100 Digital Signal Microprocessor, 2K of data memory 16 bits wide, 2K of 24 bits wide program memory instruction space, and 2K of 16 bits wide program memory data space. This is as the board comes. The board has been configured to allow further enhancements of memory area to a maximum of 32K program memory and 16K of data memory. We were using the board with 8K of data memory.

In the board there is a bidirectional coder/decoder (CODEC) channel and an undedicated 12 bit linear digital to analog converter (DAC) to allow for the processing of real time signals. A prototiping expansion connector is also there which allows the user to add custom made circuits for specific uses. In addition this there are four BNC connectors available so that the board can be interfaced to external instruments. Especially for the users of speech processing applications the features of interest are, a microphone jack, input preamplifier speaker jack, and an output amplifier.

The Evaluation board's ADSP 2100 runs under the control of an on board host processor (the host microprocessor used here is the Intel 8088) enabling one to access debugging tools. The evaluation board is interfaced to an external host computer system (in our case a PC) which runs the evaluation board's software and the ADSP 2100's software, the Evaluation Board serves as a real time development tool. The Evaluation Board is connected to a terminal and host computer via two RS–232C serial connectors

The evaluation board block diagram is shown in Figure 2.4.

Evaluation Board supports three execution modes:

- Emulator mode : In this mode the Evaluation Board runs at the full processor speed.

- Extend mode : In this mode the system updates the screen showing various displays every cycle during the program execution.

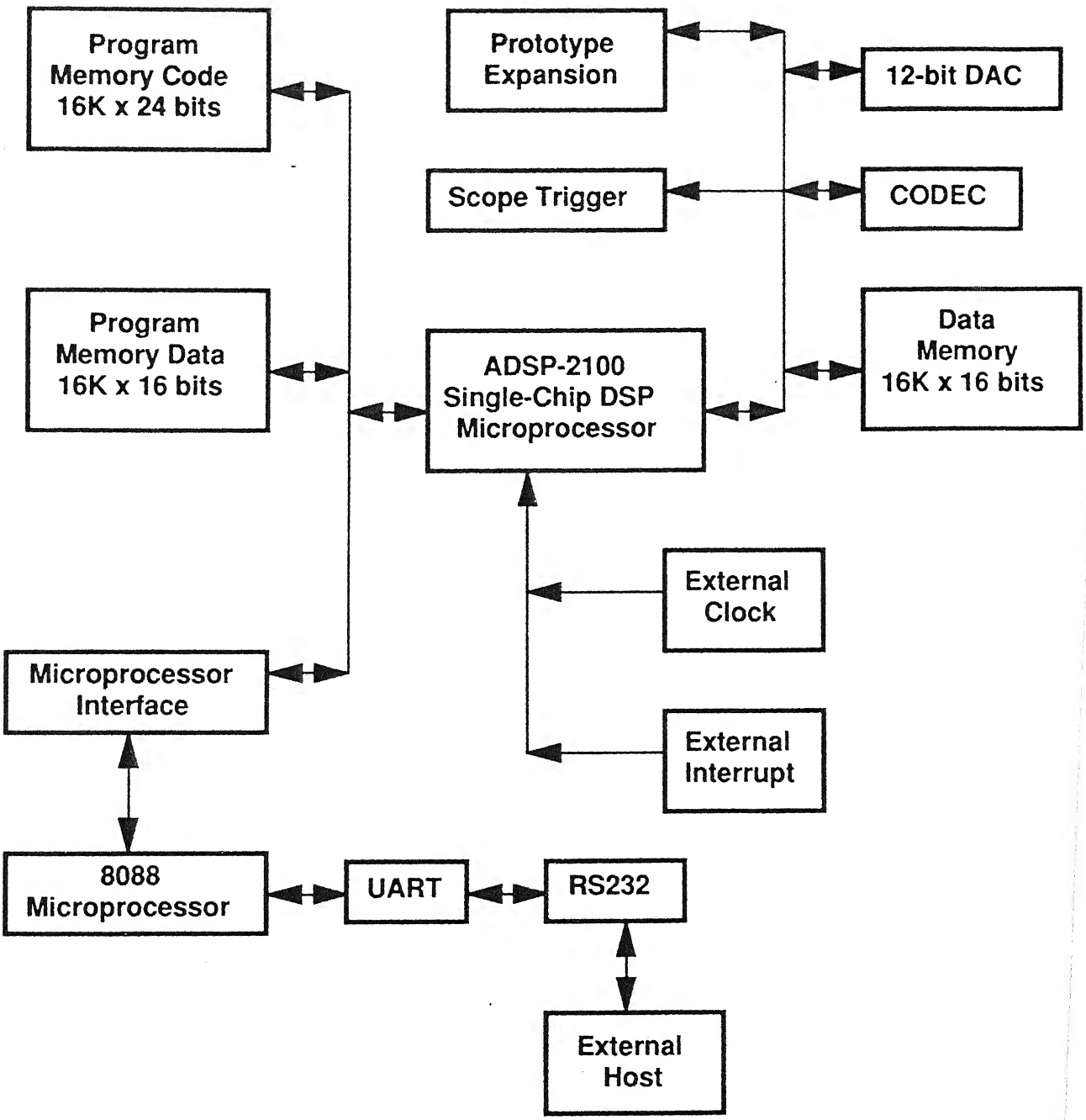


Figure 2.4 Evaluation Board Block Diagram

• Single–Step mode : In this mode the processor is made to execute a single instruction per RUN command issued from keyboard.

Evaluation Board software has four basic display modes:

• Register display : In this mode the screen displays the contents of ADSP 2100's primary and alternate registers.

• Program memory display : In this mode the screen displays contents of the program memory.

• Data memory display : In this mode the screen displays contents of data memory.

• Stack display : In this mode the screen displays contents of ADSP 2100's program counter stack.

Evaluation board software allows one to:

• Download ADSP 2100 software code from the host computer where it is generated using the Assembler, Builder and Linker.

• Modify the contents of registers, program memory, data memory and program counter.

• Set break points in program memory.

• Display user defined addresses or values symbolically.

• Patch, Delete and execute code.

The Evaluation Board to run requires an external $\pm$ 12 V and $\pm$ 5 V power supply. A list of evaluation board commands is given in Appendix F

2.4.3 Running programs on the Simulator and Evaluation board

After we have all the required files (mainly the Executable file, and Architecture Description file), we have to down load these files to the system on which we are testing our program. We discuss down loading programs both on to the Simulator and the Evaluation board from the PC.

The simulator reads the Architecture Description file (.ACH), Memory Image file (.EXE) and Symbol Table file (.SYM). The Architecture description file is input to the simulator to start an ADSP 2100 simulation session by configuring the simulated system as per target architecture. Here we just mention how the Simulator is invoked and the test program loaded and run.

The Simulator is invoked from the host system using the command:

DSPPS filename[.ach] <cr>

if the extension is not given it defaults to .ACH. After this the Simulator screen appears in its register display mode. We then load the Memory Image file (.EXE) and the Symbol Table file (.SYM) when the Simulator prompts with a '>' . These are loaded by typing.

LO filename

This commands tells the simulator to load the .EXE and the .SYM files. We then use the Read image or other Simulator instructions to load the data files which have been created as per the requirements with proper header and terminations. Then the program counter is set to the address location from where the program begins and RUN command is issued at the prompt. When the Simulator halts the program is verified.

Now we come the Evaluation Board. First of all we should have on the PC the communication software PC–VT. We then simply issue the command PC–VT. This links the Evaluation Board to the PC and shows the register contents of the ADSP 2100 microprocessor (i.e it is in the register display mode). When the Evaluation Board issues a prompt '>' then the commands are given from the key board. To down load the files we have to first go into the transparent mode. Using the command "<ctrl> D" we go into the transparent mode. Then press "<ctrl> F4" and transmit the Architecture Description file (.ACH) along with proper terminations. Similarily we transmit our other files i.e .EXE and .SYM files. If there is any data file to be transmitted they are also transmitted (here we must be careful to note that the data file has the appropriate header and terminations, so

that it goes to the proper memory location). Then we come out of the transparent mode using "<ctrl> D". The program counter (PC) is set to the address from where the program execution has to start by using the appropriate commands. Then the RUN command is given. After the Evaluation Board stops execution we can see the data in the memory locations and the registers.

SUMMARY

In this chapter we reviewed the ADSP 2100 microprocessor and various other associated development tools. Now that we are aware of the features of the ADSP 2100 DSP microprocessor, Assembler, System Builder, Linker, Simulator, Evaluation Board, we are now in a position to start working with these tools at our disposal, for the development of a speech processing system.

CHAPTER THREE

SPEECH BANDWIDTH COMPRESSION TECHNIQUES

3.1 INTRODUCTION [4,5]

The main force behind the requirement to reduce bandwidth necessary for communication, has always been economic pressures. Human speech (which is just another method of communication) has not escaped these forces. The development of efficient modulation schemes and other important factors, such as privacy tend to favor digital coding strategies in present day communication systems. Figure 3.1 shows the spectrum of the transmission rates of speech. The region above 5000 bits/sec belongs to a family of techniques known as waveform coders. The general strategy of tackling the development of such waveform coders is to track the time domain speech waveform in different ways such that, they tend to become simpler and less dependent upon the particulars of speech production as the allowable rate for transmission increases. Of the different ways extremely high quality can be achieved by simple straightforward digitization (called Pulse Code Modulation (PCM) coding) of speech waveforms if, 64 Kbits/sec or more of bandwidth is available. Techniques which are used at the lower end of this region of transmission generally exploit some knowledge of speech production process and adaptive tracking philosophies are frequently preferred such techniques are called Adaptive PCM (ADPCM).

For transmission rates below 5000 bits/sec it is accepted that, it is necessary to use some sort of parametric modeling of speech production process to reduce transmission bandwidth further. Due to this modeling the resultant quality of speech detoriates and is typically of a somewhat synthetic nature. For this purpose a speech source model of limited complexity is hypothesized, and only the necessary information needed to

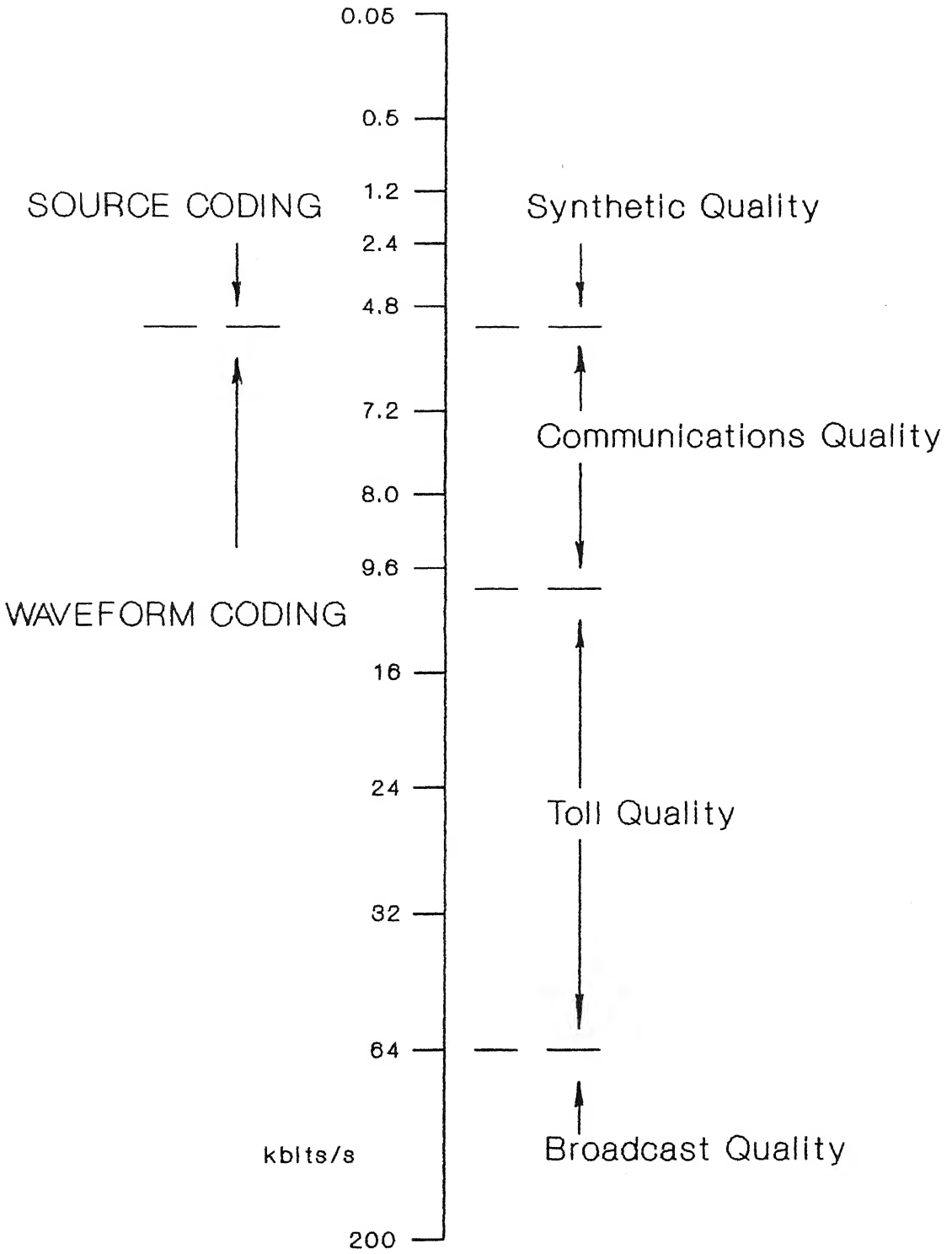


Figure 3.1 Spectrum of Speech Transmission Rates

characterize the model is extracted for the purpose of coding and transmission. At very low rates of transmission, say below 100 bits/sec, it becomes necessary to model speech as if it were comprised of a finite set of elemental sounds, and the combination of these elemental sounds is governed by a set of strict set of rules. These set of rules and constraints are derived from linguistic and physiological considerations. For an utterance to be modeled correctly it is necessary to identify the appropriate sequences of elemental sounds and transmit symbols, and identifying each one uniquely.

In this chapter we discuss a canonic narrowband speech processing model, methods used to calculate the pitch of the sound samples, and the various voice bandwidth compression techniques in use today. Channel Vocoders and Linear Predictive Vocoders (LPC), are the most in use today and they take up a major share of currently deployed vocoding equipment worldwide. The other two methods are Homomorphic Deconvolution and Formant Tracking Techniques but these are still very much in the developmental stages.

3.2 CANONIC NARROW BAND MODEL [7]

Before taking up the discussion on the vocoder structures, it would be helpful to briefly review the essentials of speech production process and point out significant features of the resultant waveforms. Given some knowledge of the mechanisms of the speech production process, it is possible to define a generalized engineering model of speech production which helps to unify various narrow band vocoding techniques described later.

A simple and straight forward schematic diagram of the human speech production system and its associated organs is shown in Figure 3.2. It consists mainly of a series of cavities whose shapes and interrelationships can be time varied by moving the jaw, tongue, lips, and velum in accordance with the voluntary control of the speaker. The cavities are made to assume various acoustic resonance patterns as a function of

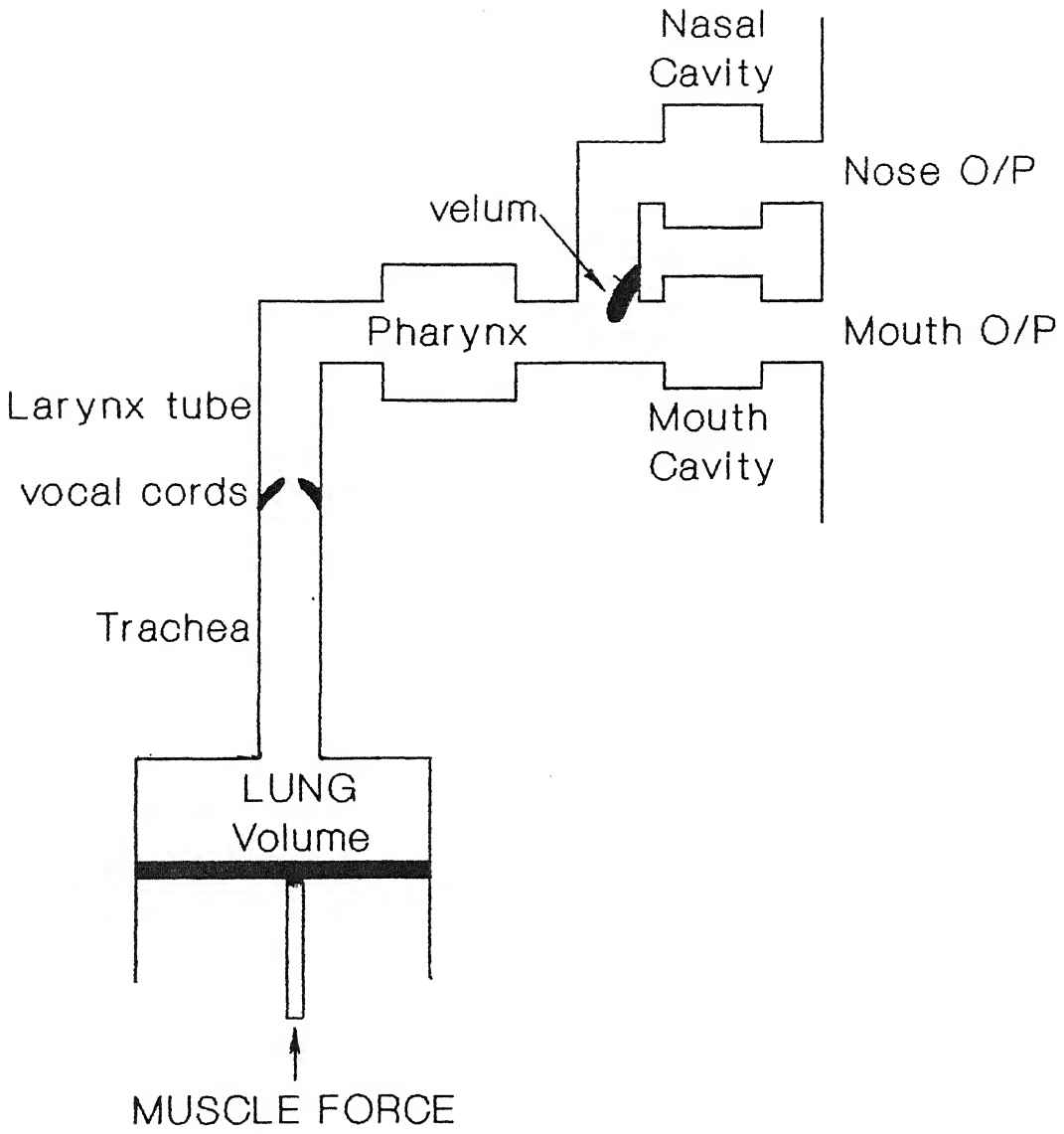


Figure 3.2 Schematic of Human Speech Production System

configuration which give rise to the various unique speech sounds. The cavity complex, is referred to as the vocal tract, is excited by puffs of air expelled by the lungs through the vibrating vocal cords of the larynx. The output sound pressure waves are radiated from the lips, nose, or a combination of the two. Sounds can also be produced by relaxing the vocal chords and allowing an unmodulated flow of air to escape the vocal tract. Specific types of turbulence introduced by particular tongue, teeth, and lip configurations impart the unique quality necessary to distinguish the several sounds of this type from one another.

Consider the Figure 3.3. It shows some typical speech wave form segments. In some waveforms there are clear evidences of periods when the waveform is nicely periodic, and so a well defined impulse response for these segments can be identified. These intervals correspond to times when the vocal tract is in a fixed configuration and the vocal cords are vibrating. In such cases the vocal tract is said to be in a "voiced state" and vowel like sounds are typically produced. There are also periods when the waveform is irregular and noise like. These are instances during which the vocal cords are inoperative and the vocal tract is in the "unvoiced state". Hiss like fricative sounds are produced in this way. If a spectrum analysis were to be performed on the speech segments corresponding to the stable vowel sound, the result would be much like that of Figure 3.4. The production of such sounds can be considered to be the result of two contributions: a slowly varying underlying component attributable to the vocal tract and a rapidly varying comb like structure due to the periodic excitations. The vocal tract component typically features several peaks or resonances which are commonly referred to as "FORMANTS". The regularity of the comb like structure is indicative of the validity of the periodic excitation assumption over the observation interval.

The discussion above suggests a convolutional model for speech production process, where the vocal tract assumes the form of a slowly time varying linear filter, excited by a two stage source. Such a canonic model indicates an analyzer synthesizer type

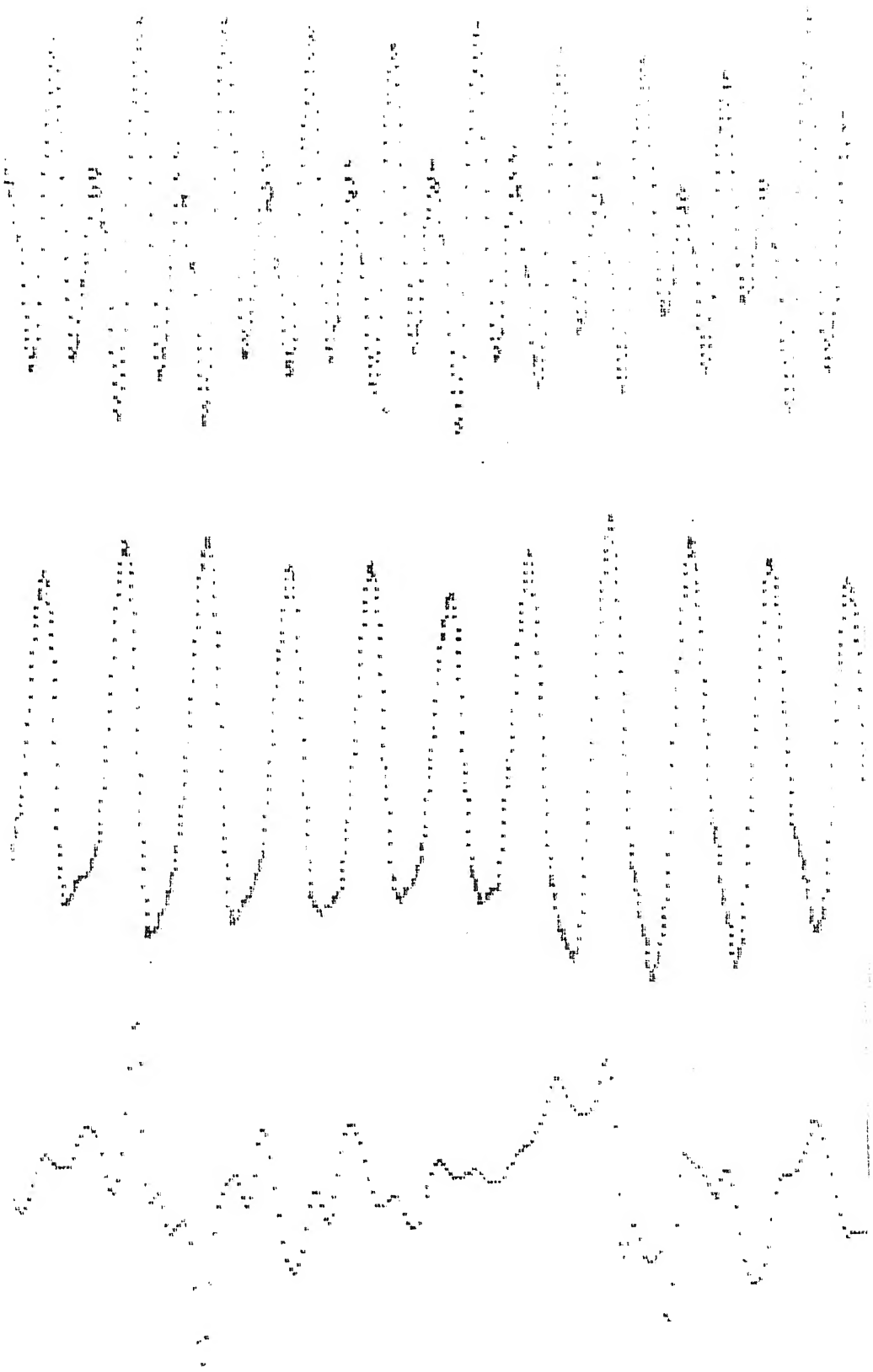


Figure 3.3 Some Typical Speech Waveforms

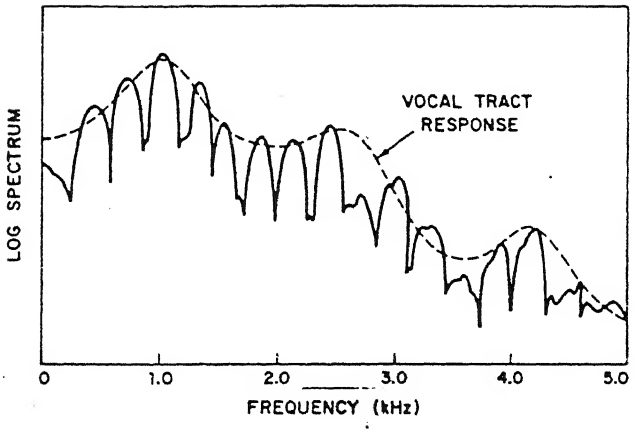


Figure 3.4 Typical Vowel Spectrum

of as structure shown in Figure 3.5. The synthesizer at the receiving end assumes the form of generalized voice production idealization. The exciter acts as an impulse generator during voiced intervals or a noise source during unvoiced sounds. To operate the synthesizer it is necessary to provide periodic updates of the vocal tract control parameters and excitation information. For the human ear an update rate of around 50 times/sec is enough although some information bearing signals may have to be tracked at a much higher rate.

The control parameters for the transmission and synthesizer are extracted at the transmitter end by the analyzer part of the vocoder system. Here the input speech is segmented into finite length observation intervals called "FRAMES" and an attempt is made to find in some sense the best average characterization of the vocal tract and the excitation source over this time span. Two major subsystems that are involved in the characterization are : the vocal tract parameter estimator and the excitation determination, more commonly termed the "pitch extractor". The major differences among the four vocoding techniques to be discussed arise from the particulars of the vocal tract characterization posed and the associated excitation process.

3.3 PITCH EXTRACTION

We will first overview the two different ways possible to determine the pitch. The two ways possible are classified as the time domain method and the frequency domain method.

The first step in the process of pitch extraction usually involves special filtering of the speech signal followed by some relatively computationally intensive preprocessing which is done to extract raw measures of potential periodicity in the speech signal. The next step usually involves reduction of the raw measurement data to yield an initial best guess of the waveform period over the given frame. The final step involves the

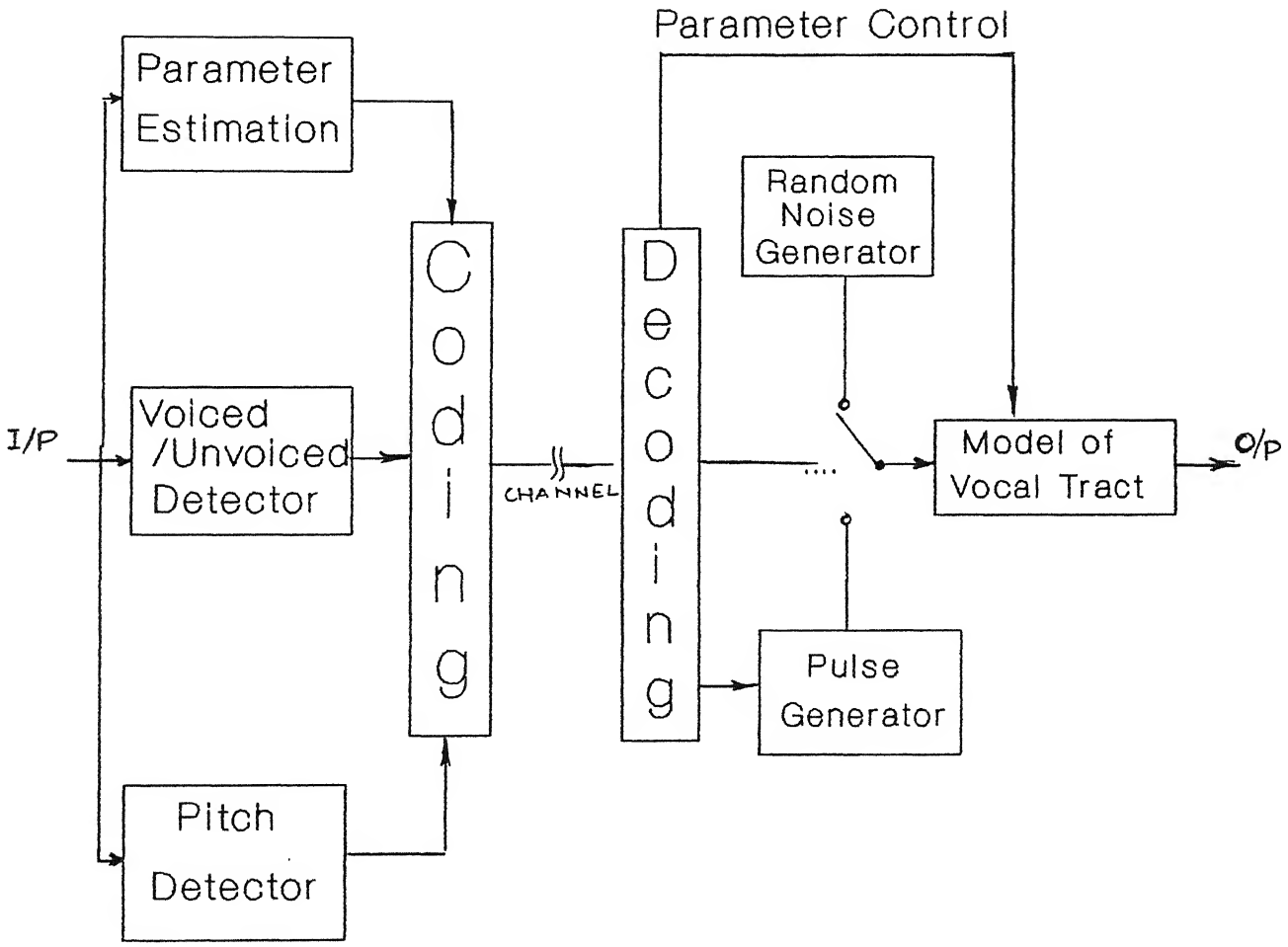


Figure 3.5 Canonic Narrowband Vocoder

application of a set of heuristically determined constraints and editing rules which tend to modify or fully reject pitch and voicing decisions.

The processing can be tackled in either the time or frequency domain. In the time domain the best method is to measure the elapsed time between suitably processed waveform events related to peaks and valleys in the low pass filtered signal. A parallel threshold processing philosophy is used to enhance the generality and robustness of the periodicity measure, and a variety of sophisticated coincidence measurements and derive the best overall estimate. Other time domain approaches necessitate calculation of the autocorrelation function of the speech input. The presence or absence of periodicity is inferred by the examination of secondary peaks in the determinant functions and the interval between peaks is indicative of the raw period estimates. The frequency domain approaches exploit the comb like structure of the voice spectrum that we had discussed earlier in Figure 3.4.

3.4 VOCODERS

3.4.1 Channel Vocoders.

Channel vocoding is the oldest of practical narrowband voice processing techniques. It is based on the original work of Homer Dudley performed in the late 1930's. A variety of designs and configurations for this method of coding have been developed with varying degrees of success. The full specification of a channel vocoder involves a large number of degrees of freedom and its performance is a very sensitive function of the specific choices and tradeoffs made.

The basic way of modeling a channel vocoder is to attempt a direct representation of the combined vocal tract and excitation power spectrum by summing up together a set of suitably weighted frequency responses. A typical structure of a channel vocoder is shown in Figure 3.6. The synthesizer is constructed using a bank of bandpass

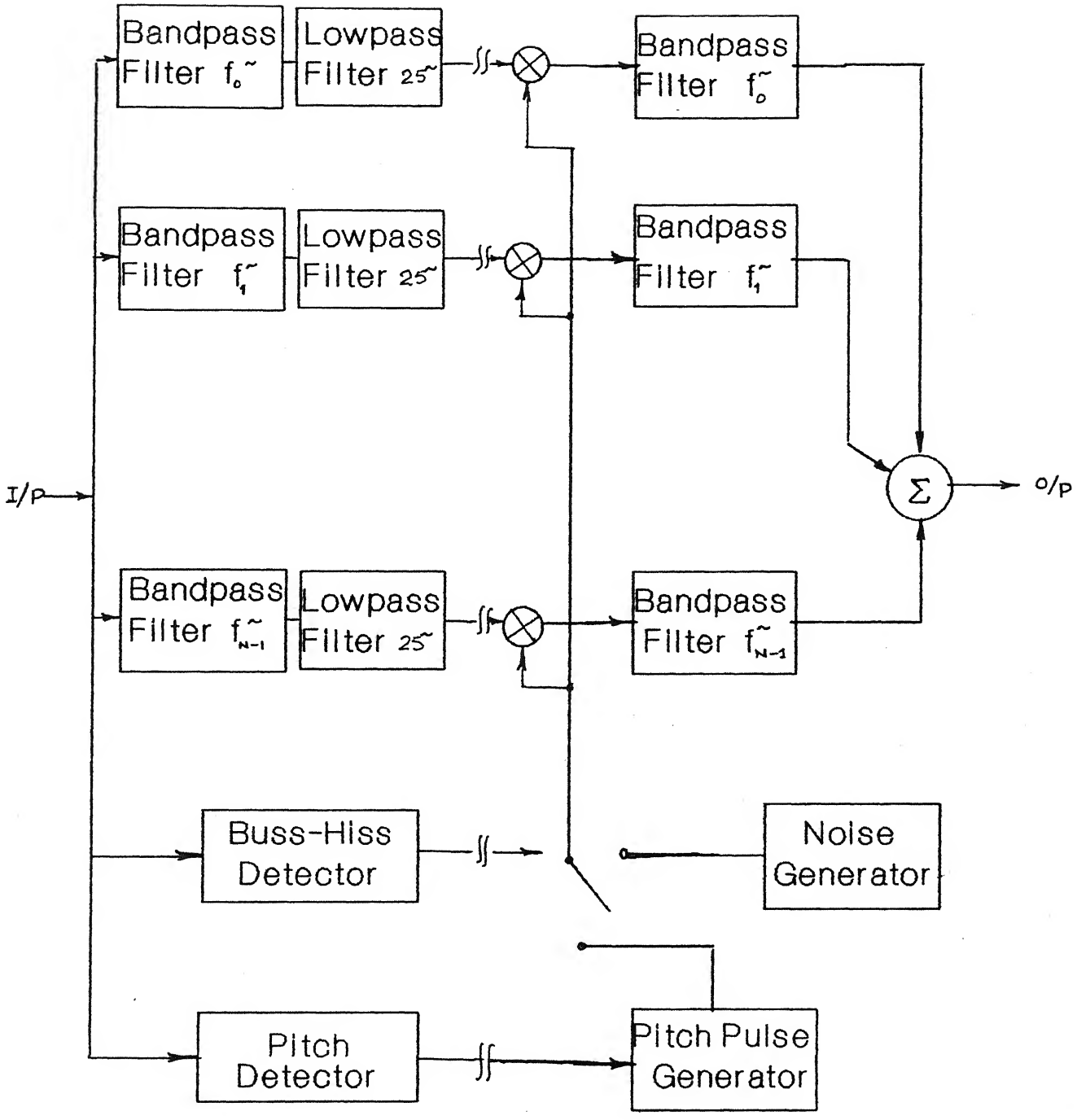


Figure 3.6 Typical Channel Vocoder

filters which are driven by a standard binary excitation source model. The input to each filter is modulated by a weighing factor which were calculated during the analysis procedure (i.e.during modeling) and it is then passed on to the synthesizer.

The modulation weights are extracted by a classic time domain analog short time spectral analysis structure comprising a parallel aggregate of bandpass filter, rectifier, lowpass filter cascades (channels). The outputs of analysis bank, which are sampled typically every 20 msec or so, represent the estimates of speech waveform power spectral magnitude at a set of discrete points in frequency. Although a number of methods have been devised, the most practical approach to transmitting the modulation weights involves logarithmic digitization followed by a suitable transformation to further reduce the number of bits necessary in characterizing the voice spectrum.

3.4.2 LPC Vocoders

Channel vocoders are the products of analog era. The family of vocoding techniques based on Linear Prediction (LPC) are the product of the digital era. The primary motivating force behind the development of LPC as a vocoding technique are the cold hard realities of cost and performance.

The basic fundamentals on which the LPC systems are based is that the vocal tract can be modeled as an all pole filter whose discrete time transfer function is of the form:

$$H(z)=\frac{G}{(1-\sum_{k=1}^{p} a_k z^{-k})} \qquad (3.1)$$

where p is the order of the system.

It is the function of the analyzer to provide frame by frame estimates of the polynomial coefficients a_k's and G by the processing of a finite and possibly overlapped

segments of input speech signals. LPC can be viewed as a spectral fitting problem subject to a type of least squares optimality criterion. In spirit an attempt is made to separate or deconvolve the effects of the vocal tract and the excitation source. Depending on the precise formulation, one solution to the problem can be expressed in the linear matrix form:

$$\underline{a}^{p} = -\underline{R}^{-1}\underline{r} \qquad (3.2)$$

where the p x p $\underline{R}$ matrix is Toeplitz and comprised of the first p auto correlation coefficients of the input speech signal segment. The p^{th} order column vector $\underline{r}$ consists of the second through $p+1^{st}$ auto correlation coefficients implying that the first p+1 coefficients are necessary to compute the entire solution. This particular formulation is termed as the auto correlation method and it is amenable to expedient solution through various recursive techniques such as the Levinson Durbin recursion due to the Toeplitz nature of $\underline{R}$. Though there are other analysis approaches which do not give rise to a Toeplitz $\underline{R}$ yet have their own unique advantages, but we note that the auto correlation solution always corresponds to a stable, causal system in the absence of finite arithmetic errors.

The vocal tract can be modeled as a series of lossless, equal length tubes as shown in Figure 3.7. If we do so the wave propagation equation can be manipulated in a such a way so as to yield an electrical analog as shown in Figure 3.8. This regular lattice type filter structure is termed an acoustic tube realization and the K parameters characterizing the tube are simple functions involving ratios of adjacent lossless tube cross sectional areas.

3.4.3 Homomorphic Deconvolution

Processing using LPC coefficients can be considered to be a type of spectral smoothing. Our main aim is to filter out the periodic excitation infrastructure and bring out the characteristics of the slowly varying vocal tract contribution. Homomorphic

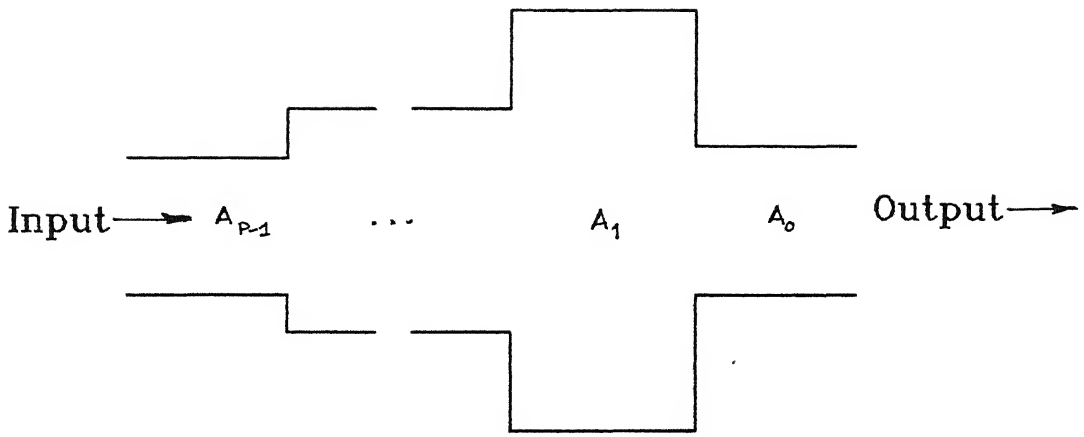


Figure 3.7 Physical Lossless Acoustic Tube

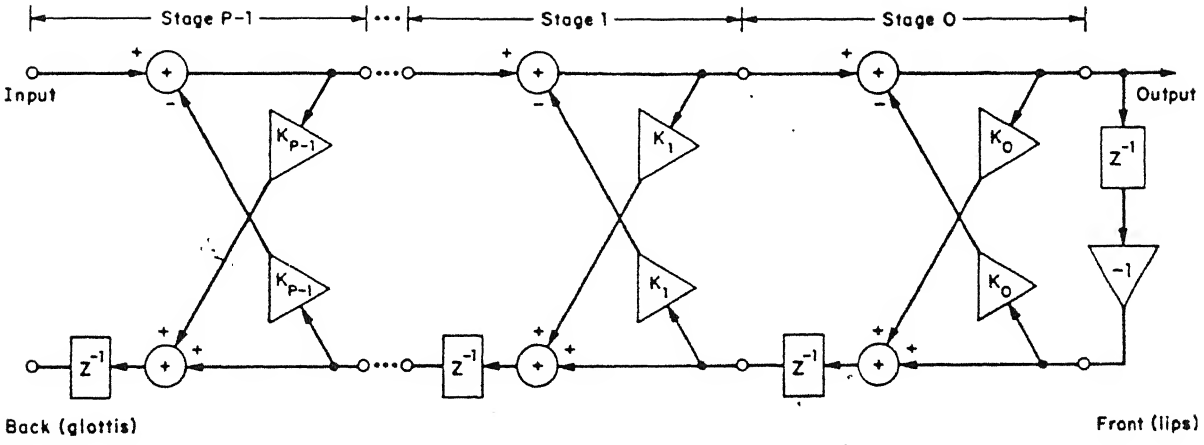


Figure 3.8 Digital Filter Representation of Acoustic Tube

deconvolution is really an alternative approach to the same style of processing. This is a specialization of the generalized theory of linear filtering developed by Oppenheim. Here a basically non–linear problem is reformulated in linear terms through a suitable set of transformations. In the case of speech, the spectral contributions due to vocal tract and excitation effects are presumably combined as a product of factors if the convolutional model holds. A logarithmic transformation of the composite speech spectrum will be comprised of strictly additive terms, and the separation can be mechanized via linear filtering methods. Due to mathematical complexities, phase is discarded and the logarithmic transformations in practical systems is applied to the spectral magnitude only. It can be shown that this simplification implies either a fixed or minimum phase model of vocal tract filter and hence there is no all pole restriction. The actual vocal tract characterization ultimately assumes the form of an impulse response which is derived from direct spectral estimation.

The question of designing the log spectral smoothing filter now arises. It has been that an arbitrary non–recursive filter can be realized via a Discrete Fourier Transform (DFT), a weighing function multiply, and an inverse DFT. Considering the graph of spectrum versus frequency (Figure 3.4) it is reasonable to expect the slowly varying vocal tract contributions to lie near the origin and rapidly varying excitation effects to appear in the transform. The transform function has been given the name CEPSTRUM and has its own interesting properties. At present it is necessary only to realize that by weighing the portion of the CEPSTRUM near the origin using some type of idealized low pass filter, function vocal tract effects can be effectively isolated from those due to excitation. The shape and cutoff chosen for the weighing function (determined emperically) uniquely defined the desired log spectral smoothing filter. The "high pass" regions of the CEPSTRUM can be used directly as a basic pitch period determinant by way of its presumed periodic structure. Figure 3.9 shows a classic homomorphic analysis synthesis

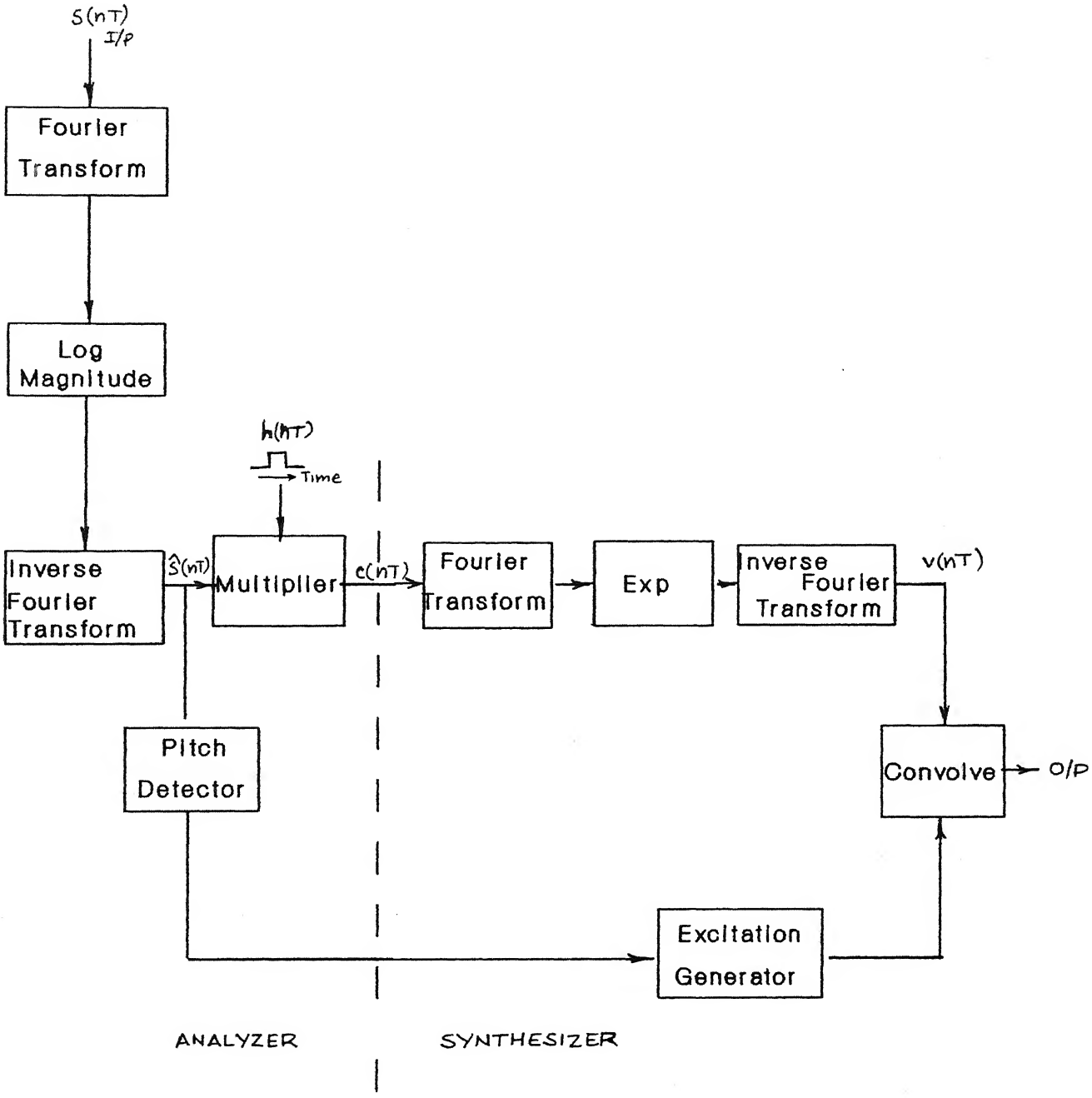


Figure3.9 Typical Homomorphic Vocoder

system. A kind of vocoder called Spectral Envelope Estimation (SEE) vocoder overcomes certain difficulties that are encountered in the conventional vocoder.

3.4.4 Formant Tracking

It was indicated in the LPC discussion that the vocal tract can be reasonably modeled for vowel sounds as a time varying, all pole filter and the acoustic tube analogy affords some physiological justification for this point of view. Although a lattice network characterized by K parameters is one acceptable topology, it is well known that any transfer function can be equally well represented as either a cascade or parallel connection of resonators via factorisation or partial fraction expansion respectively. The cascade form bears the most intuitively obvious relation to the mechanical concept of vocal tract and requires pole location data for each resonator and an overall gain factor for complete characterization. In practical situations the number of resonator sections is restricted to three or four (corresponding to typically observed formant structure) thereby further economizing on the amount of control information required. However not all speech sounds can be satisfactorily modeled with such an arrangement. For example nasal sounds imply zeros in the transfer function introduced by the addition of a branch (i.e. the nasal cavity) to the basic acoustic tube model. Also certain consonant sounds require special handling and sundry additional artifices are required to patch up the model.

The parallel formulation requires inherently more control information and includes a gain factor as well as a pole location descriptor for each resonator. However, an error in deriving the control signals for a given section tends to disrupt only a localized band of frequencies in the synthetic spectrum rather than propagate anomalies throughout the entire spectrum as would be the case in a cascade structure. Also, by adding an extra suitably defined section or two it is possible to model more conveniently the effect of zeros appearing in critical regions of vocal tract transfer function, such as during nasalization,

and accommodate consonant sounds. The parallel structure has been shown to be capable of producing very high, quality, natural sounding speech when properly controlled. For these various reasons the parallel structure is usually selected as the basic synthesis element in the current formant tracking research efforts.

A typical formant synthesizer incorporating several detailed features of voice production process is shown in Figure 3.10. The variable resonator system may be comprised of three or four fully flexible sections and perhaps one other of restricted flexibility used in producing consonant sounds. Usually it is not necessary in practice to dynamically control formant bandwidths although provisions can be made to couple bandwidth to control signal variations via a prescribed set of fixed rules at the synthesizer. An extra wideband filter may be included to model the upper frequency regions for unvoiced sounds and an overall fixed compensation can be incorporated to account for radiation effects from the lips. Provisions may be also made for a more general, mixed source type of excitation where the relative proportions of periodic and noise components are variable. It has also been found that a properly tailored waveform shape is preferable in terms of producing natural sounding speech to a simple impulse as a voiced excitation component. Since the parallel formant synthesizer is basically a programmable array of recursive filters and as such should be amenable to implementation via sampled analog VLSI techniques, the potential for efficient hardware realization appear high.

The analysis problem, or deconvolution of the necessary synthesizer control functions in an automated, real time way, is an imposing one and a continuing subject of active research. Most approaches begin with some type of short time spectral analysis based on filter bank structures, LPC related methods, or DFT techniques. The basic idea is to search the spectral magnitude estimate for a set of localized maxima in the hope that formant and amplitude correspondences can be identified. The raw data, or initial guesses, may be refined in a variety of ways to rule out unlikely alternatives and use is made of

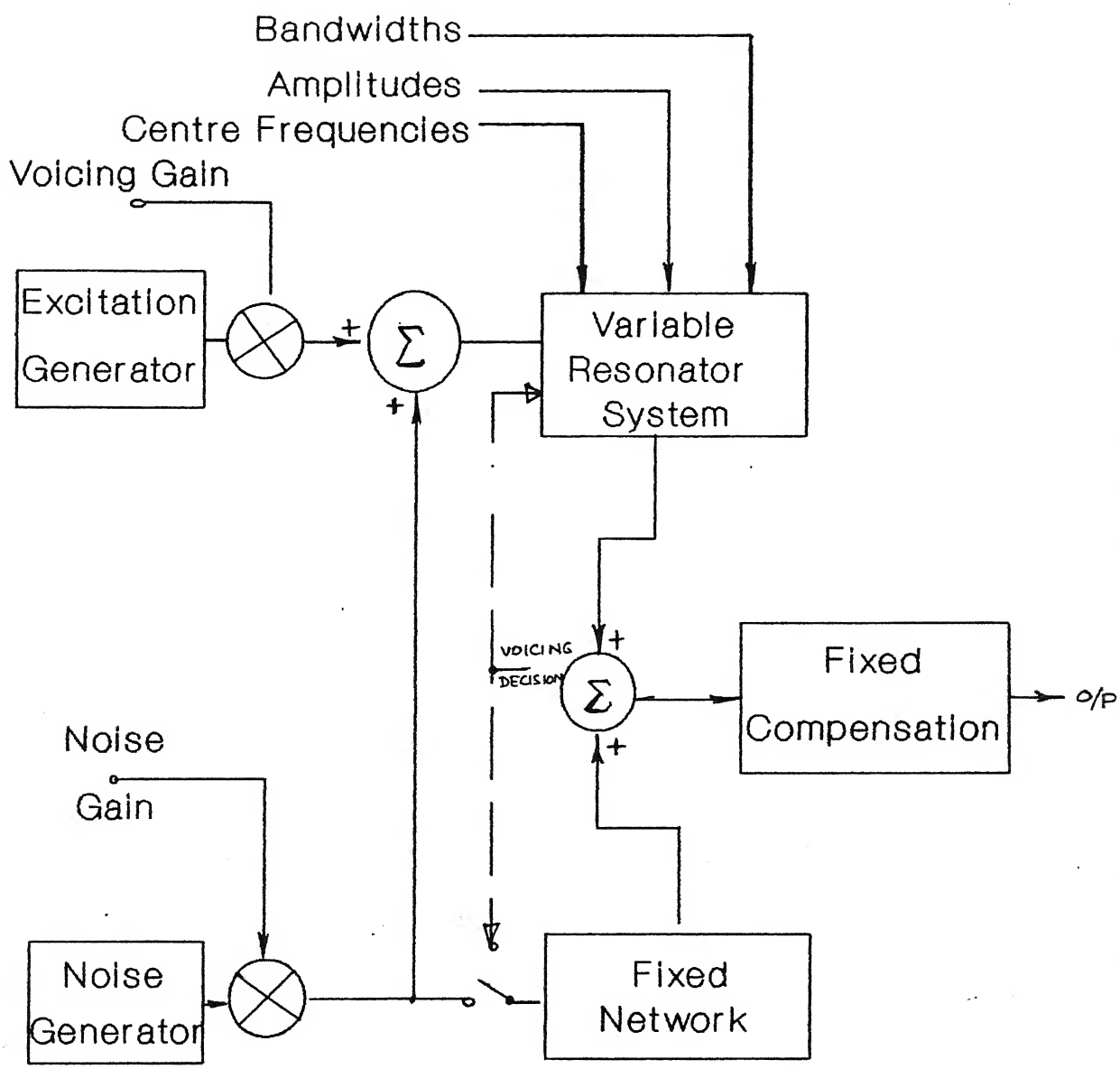


Figure 3.10 Typical Formant Synthesizer

previous history (hence the term tracking) as well as known physical and linguistic constraints. There are also issues relating to the rate at which control information should be extracted. During periods of relative stationary, such as occur during sustained vowels, 40 to 50 Hz would seem sufficient. However it is well known that a great deal of intelligibility cues are contained in rapid transitions between vowel and consonant sounds thus implying peak analysis rates as high as 100 to 200 Hz.

The most sophisticated of analysis techniques under investigation exploits a variable frame rate (with an average value of about 1200 bits/sec) and employs an iterative analysis by synthesis methodology. Via this approach, formant position and amplitude data are successfully refined by comparing the synthetic spectrum (as produced by the parallel synthesizer) at each stage with the input speech spectrum. A best fit is established by virtue of an appropriate optimality criterion. This type of system is capable of extremely high quality and natural sounding speech but remains quite experimental. The analysis algorithm are relatively computationally intensive

SUMMARY

The channel vocoder is the oldest of speech bandwidth compression techniques and has been in wide spread use for more than forty years. Technical advances in the area of sampled data analog VLSI may well rescue this venerable performer from immediate obsolescence through drastically reduced hardware complexity and power dissipation. Modern all digital techniques, as typified by LPC type vocoders, would seem to represent the most likely successor to the channel vocoder as a relatively universally accepted government and military standard. The most sophisticated of such systems have demonstrated intelligibility and quality slightly beyond that of the best channel vocoders under selected benign environmental conditions. Given the rapidly advancing state of digital LSI technology, future hardware realizations can be expected to compete favorably

with state of the art channel vocoder designs in cost, size and complexity. The spectral estimation and formant tracking approaches are still evolutionary and relatively immature in terms of practical hardware development. They do offer the promise of considerably higher quality than presently possible at transmission rates below 2400 bits/sec in the near future.

In this chapter we have discussed first of all the model of the speech production process. Pitch extraction philosophy and the major voice compression techniques.

CHAPTER FOUR

CODE EXCITED LINEAR PREDICTION OF SPEECH

4.1 INTRODUCTION

Present day digital hardware equipment make it a good proposition to use LPC for the purpose of speech bandwidth compression. To develop our codec we used code excited linear prediction of speech. In this chapter we study the CELP algorithm and its speed up mechanisms. We also discuss the implementation on the ADSP 2100 processor.

The original code excited linear predictive coding (CELP) algorithm was introduced in 1984. It main achievement is that it is able to provided good speech quality at intermediate bit rates, but the vector quantization of its excitation signal is of very high complexity i.e. the computational effort required for its determination is very high. CELP vector quantization requires dynamic weighing of the vectors. This prevents the use of conventional speed up procedures.

4.2 THE CELP ALGORITHM

Speech signal is sampled and divided into units called frames. CELP coders operate on these frames sequentially, one frame at a time. A filter is used to describe the spectral envelope of the speech signal. The coefficients of this filter are obtained using the linear prediction (LP) analysis technique. They are quantized so that the same filter can be constructed at both the transmit and receive end of the channel. The excitation for the filter is determined by an analysis–by–synthesis procedure. In this procedure a set of candidate excitation sequences is stored in a codebook. Synthetic speech is generated using each of these sequences. The index of the sequence producing the most accurate speech is

transmitted to the receive end of the channel, where this codebook is also stored. Using these filter coefficients and the excitation vector speech can be regenerated at the receive end of the channel.

We have to encode the excitation signal. To do this it is useful to first compute the target excitation sequence 't' for the current frame being processed. The target excitation sequence is defined as that vector which will drive the synthesis LP filter to produce the current frame speech signal.

The filtering operation of the excitation sequence by the all pole filter can be performed by the convolution of the excitation sequence 't' with the impulse response of the filter. For a single frame, this convolution can be written as the matrix multiplication 'Ht'. Here H is an N by N lower triangular Toeplitz matrix containing the impulse response $\{h_i\}$ of the filter in its first column.

$$H = \begin{bmatrix} h_0 & 0 & & & 0 \\ h_1 & h_0 & & & \\ \cdot & h_1 & & & \\ \cdot & \cdot & \cdots & & \cdot \\ \cdot & \cdot & \cdots & h_0 & 0 \\ h_{N-1} & h_{N-2} & & h_1 & h_0 \end{bmatrix} \qquad (4.1)$$

If 'z' is the zero–input response of the synthesis filter for the current frame, then the speech signal s can then be written as:

$$s = H.t + z \qquad (4.2)$$

There are different algorithms to quantize the target vector 't'. Some algorithms quantize the vector by the sequential determination of a set of pulse locations and amplitudes. In contrast the CELP algorithm encodes the target vector via a shape gain vector quantization. The codebook used in this vector quantization can consist of stochastic

or deterministic sequences or sequences obtained from a training procedure.

After the impulse response of the filter is obtained it is required to determine the excitation vector for the filter. To obtain the excitation vector an exhaustive search is performed. In the search, synthetic speech vectors are generated by each of the set of optimally scaled candidate vectors $\mu^{(i)}x^{(i)}$.

To evaluate the accuracy of the synthetic speech signal the least squares error criterion is used. The selection of the best candidate vector for generation of speech is viewed as a shape gain vector quantization of the target excitation vector. This shape gain vector quantization uses the following dynamic error criterion.

$$\epsilon^{(i)} = (t-\mu^{(i)}x^{(i)})^T H^T H (t-\mu^{(i)}x^{(i)}) \quad (4.3)$$

where $\mu^{(i)}$ the optimal scaling factor for the candidate vector $x^{(i)}$. $\mu^{(i)}$ is defined as

$$\mu^{(i)} = \frac{t^T H^T H x^{(i)}}{x^{(i)T} H^T H x^{(i)}} \quad (4.4)$$

This expression for the optimal scaling factor $\mu^{(i)}$ is substituted into equation (4.3), and the error value for a particular candidate excitation vector obtained. The vector giving the least error, is the excitation vector for the filter, to generate current frame speech signal at the receive end of the channel.

The 'H' matrix in (4.1) results in an error criterion with a symmetric weighing matrix H^TH. This is very advantageous as symmetry in the matrices helps in the development of fast algorithms (it is more helpful if the error criterion is as in (4.3)). The spectral weighing matrix H^TH becomes a Toeplitz matrix if the impulse response $\{h_i\}$ truncated after 'R' samples, is used in its entirety for each sample of the excitation vector. In most situations the value of 'R' is chosen to be less than 'N', since the perpetual weighing reduces the effective length of the impulse response. In such cases, H^TH becomes a band matrix (in addition to being symmetric and Toeplitz). The matrix 'H' now is:

$$H = \begin{bmatrix} h_0 & 0 & \cdots & & 0 \\ h_1 & h_0 & \cdots & \cdots & \\ \cdot & \cdot & \cdots & \cdots & \cdot \\ h_{R-1} & h_{R-2} & \cdots & \cdots & \cdot \\ 0 & h_{R-1} & \cdots & \cdots & \cdot \\ \cdot & \cdots & \cdots & \cdots & h_0 \\ & & & & h_1 \\ \cdot & \cdots & & \cdots & \cdot \\ \cdot & \cdots & & h_{R-1} & h_{R-2} \\ \cdot & \cdots & & 0 & h_{R-1} \end{bmatrix} \qquad (4.5)$$

The error criterion in (4.3) which uses the H matrix of (4.1) is termed the covariance approach, while the modified criterion which uses H matrix of (4.5) is termed the autocorrelation approach. The advantage of the autocorrelation approach is that the autocorrelation error criterion results in a more equitable weighing of samples of the excitation vector. There is not much difference in the performance of the two error criterion, but the major advantage of using the error criterion with H matrix of (4.5) is the symmetry created in H^TH matrix, which now is a symmetric Toeplitz matrix; and if $R < N$, a symmetric Toeplitz band matrix. In contrast, if (4.1) is used in the error criterion, then H^TH matrix is only a symmetric band matrix.

4.3 SHORT DESCRIPTION OF CERTAIN FAST ALGORITHMS

In the following section we discuss the properties of methods which minimize the computational effort of the CELP algorithm. The algorithms are discussed in short. Algorithms are classified into several groups: fast procedures which rely on specially designed codebooks, various multistage search procedures, procedures operating in

transform space, the autocorrelation method, methods which use overlapping codebook entries, and methods which perform the energy term calculation offline.

4.3.1 Special Codebook Designs

Our aim is to reduce the computational effort in the determination of the excitation vector. A simple but effective method to reduce the computational effort is by center clipping the candidate vectors of a stochastic codebook. Center clipping results in significant savings in the complexity if the vector $Hx^{(i)}$ is computed by adding the responses due to each of the samples of $x^{(i)}$ rather than by computing the output vector $y^{(i)}=H\ x^{(i)}$ one sample at a time. Using the former procedure, the entire column of 'H' is skipped whenever a sample of $x^{(i)}$ is zero. This results in a lot of saving in the computational effort.

4.3.2 Multistage Searching

The major computational effort required in the CELP algorithm is due to the spectral weighing of the error. If we are ready to sacrifice little bit of performance on the part of the coder, relaxing the spectral weighing can lead to significantly faster computations. Bypassing the dynamic criterion for the selection of the excitation vector, and using a pre selection methods for the candidate vectors will lead to significant improvement performance concerning speed. There are two methods which are used for pre selection. In one method the candidate excitation vector is pre selected by performing a direct (least squares) comparison with the target excitation vector. The full search procedure, including spectral weighing, is then performed on a reduced set of candidate vectors. In the another method of pre selection the codebook vectors are filtered offline through a small set of 'P' filters representing spectral classes. During operation of the coder, the spectrum of the current frame of the speech signal is first classified as belonging

to one of the spectral classes. The corresponding vectors which have been filtered offline are then compared to the current frame speech signal. The 'K' candidate vectors which perform best under this pre selection criterion are then searched using the conventional error criterion.

4.3.3 Transform Methods

In all the above methods the result there is a modification in the numerical results of the CELP algorithm. If codebooks are modified it will lead to improved speech quality. There are various procedures to reduce the complexity of the algorithm without affecting its numerical results. Two of the most well known methods are Singular Value Decomposition (SVD) procedure and the Fourier Transform procedure. In the SVD method the error criterion is decomposed as follows:

$$\epsilon = (y - \mu^{(i)} H x^{(i)})^T (y - \mu^{(i)} H x^{(i)})$$

$$\epsilon = (U^T y - \mu^{(i)} D V x^{(i)})^T (U^T y - \mu^{(i)} D V x^{(i)})$$

where 'U' and 'V' are unitary matrices and 'D' is a diagonal matrix. In the equation above the unitary properties of 'U' have been made use of. If the statistical distribution of the samples of $x^{(i)}$ is independent, identical, and gaussian, then so is that of $Vx^{(i)}$. Now that we know this a search can be performed using the following error criterion:

$$\epsilon = (U^T y - \mu^{(i)} D u^{(i)})^T (U^T y - \mu^{(i)} D u^{(i)})$$

where u are the vectors of the stochastic codebook, which represent the transform domain excitation vectors. Similarly we have an expression for the optimal scaling factor $\mu^{(i)}$. When the best candidate vector $u^{(i)}$ has been found, the actual candidate vector can be obtained from $x^{(i)} = V^T u^{(j)}$. By modifying the notation the SVD method can also be described as an eigen value decomposition method (EVD).

There is also one method which makes use of Fourier transform. It is based on the fact that convolution in the time domain is multiplication in the frequency domain.

If we denote the Fourier transform of $x_p^{(i)}$ by $X_k^{(i)}$,h_i by H_k and t_p by T_k then we obtain the new error criterion and scaling factor as shown below:

$$\epsilon = \sum_{k=0}^{k=N+R-1} (T_k - \mu^{(i)} X_k^{(i)})^* H_k^* H_k (T_k - \mu^{(i)} X_k^{(i)})$$

$$\mu^{(i)} = \frac{\sum_{k=0}^{k=N+R-1} \mathrm{Re}\ (\ X_k^{(i)*} T_k H_k^* H_k)}{\sum_{k=0}^{k=N+R-1} X_k^{(i)*} X_k^{(i)} H_k^* H_k}$$

The main effort here is in the calculation of the crosscorrelation and energy terms. But the amount of effort required here is much less.

4.3.4 Autocorrelation Method

The basic way in which autocorrelation method attempts to minimize the complexity is by the efficient determination of the energy term. The energy term is the factor in the denominator of the optimal scaling factor. It takes cognizance of the fact that the sum of squares of the convolution of two sequences is equal to the crosscorrelation of the autocorrelations of these sequences. If the energy term is written in scalar notation, and if advantage is taken of symmetry of the autocorrelation functions, we can do the following:

$$\sum_{k=0}^{k=N+R-1} \left(\sum_{m=0}^{m=R-1} h_m x_{k-m} \right)^2 = R_0^{(h)} R_0^{(h)} + 2 \sum_{k=1}^{k=R-1} R_k^{(h)} R_k^{(x)}$$

where it is assumed that $x_m = 0$ for $m < 0$, and where $R_k^{(h)}$ is the k^{th} autocorrelation of 'h'.

If the autocorrelation sequence for all the excitation candidates are stored before hand which can be easily done (all that is required is enough memory), the calculation of the energy term is very fast. The autocorrelation procedure does not put any constraints on the type of fixed codebook used, and can also be applied to trained codebooks.

4.3.5 Methods Using Overlapping Codebook Entries

Use of overlapping codebooks provides several advantages. The first advantage we mention is that the storage requirement for such codebooks are reduced. Overlapping neighboring candidate vectors can also be related to each other by shifts and end point corrections. This offers a much faster calculation of the energy term.

4.4 COMPARISON OF THE FAST METHODS

Table 4.1 shows the comparison of the computational and storage requirements of the various fast procedures discussed. Table 4.2 shows the various methods discussed with the basic criterion used and the type of codebook used.

4.5 SELECTION OF METHOD FOR IMPLEMENTATION

After going through the Table 4.1 above we felt that the system (i.e ADSP 2100 Evaluation Board) that we have, is well suited for the Autocorrelation method. It is a method which is a good balance between the computational effort and storage requirement. It also provides for an algorithm relatively simpler than the others whereas the quality or performance is not at all hampered.

4.6 IMPLEMENTATION OF THE ALGORITHM

All the implementations on the ADSP 2100 are in the form of small modules. The development is subdivided into small units and each unit developed separately which are then finally linked together.

In the development of the codec the first module calculates the LPC coefficients. For this purpose Levinson Durbin recursion is utilised. The Levinson Durbin recursion is described by four equations as follows:

Method	Complexity	Storage
Basic	MN(N+5)/2	MN
Centre Clipping	MN(2.1+0.1N)	MN
Pre Select	MN+PN(N+5)/2	MN(P+1)
Autocorrelation	M(N+R+1)	M(N+R+1)
SVD	3MN	MN
Fourier	3M(N+R-1)	M(N+R-1)
End Correction	2M(N+LR)	N+L(M-1)

TABLE 4.1

Computational and Storage Requirements of Fast Procedures

Method	Criterion	Codebook
Basic	Covariance	general, not adaptive
Centre Clipping	Covariance	stochastic & adaptive
Pre Select	Covariance	general, not adaptive
Autocorrelation	Autocorrelation	general, not adaptive
SVD	both	stochasitc, not adaptive
Fourier	Autocorrelation	general, not adaptive
End Correction	covariance	adaptive, overlap structure

TABLE 4.2

Fast Procedures, Criterion and the Type of Codebook used

$$k_i' = [r_s(i) - \sum_{j=1}^{j-1} a_j r_s(i-j)] \div E$$

$$a_i' = k_i'$$

$$a_j' = a_j - k_i' a_{i-j} \quad \text{for } j=1 \text{ to } j-1$$

$$E' = (1-(k_i')^2)E$$

where k_i are negatives of the reflection coefficients in LPC synthesis filter, a_i are coefficients used to predict next value, $r_s(i)$ are the autocorrelated values of the input, and E is the average squared error.

The autocorrelation function for input signal s(n) it defined as

$$r_s(k) = \sum_{m=0}^{m=N-1-k} s(m)s(m+k)$$

Using the above recursive equations the first module is written.

In the autocorrelation method we require the autocorrelated values of the candidate excitation vectors to calculate the energy term of the scaling factor. A separate module finds the autocorrelated vector of the candidate excitation vectors. To calculate the numerator part of the scaling factor ($t^T H^T H x^{(i)}$), there is a third module. Another module implements the calculation of the denominator term ($x^{(i)T} H^T H x^{(i)}$) in the scaling factor using the outputs of the earlier module, which calculates the autocorrelation vectors of the candidate vectors. One module takes the inputs of the above two modules and outputs the optimal scaling values $\mu^{(i)}$. The other modules calculate the final error vector from which the candidate vector which causes the minimum error is known and is subsequently used to generate the output speech.

All the modules have been appended to give a final program which is our CODEC.

CHAPTER FIVE

POLE ZERO ANALYSIS

5.1 INTRODUCTION

Poles and zeros of a function are very important values for the analysis of a system. The basic model for speech production process and its characteristics, indicate that the vocal tract can be reasonably approximated in terms of a rational transfer function represented in terms of poles and zeros. The poles correspond to the vocal tract resonances and the zeros are introduced due to such effects as coarticulation and coupling between the vocal tract and the nasal cavity. As we have seen earlier the vocal tract has two parallel paths one through the mouth and the other through the nose. It is the tract through the nose which introduce these zeros. It has been a usual method in speech analysis and synthesis systems to represent the vocal tract characteristics in terms of an all pole model, approximating the effect of the zeros by adjustment of the bandwidth of the first formant. For more general speech analysis, as carried out, for example, within the context of automatic speech recognition, or for studies directed towards linguistic aspects of speech and psychological studies of speech production, a variety of important clues is provided by information about spectral zeros.

5.2 SPEECH RECOGNITION SYSTEMS [4,5]

For the purpose of speech recognition, we require certain digital speech processing techniques which reduce speech to certain parameters, and these are stored. When something is spoken, it is also reduced to those very parameters. These stored parameters and the parameters from spoken samples are compared. In speech recognition, the goal is to determine what word, phrase, or sentence was spoken.

The area of speech recognition is multifaceted. This area alone is subdivided, with a large number of options which must be specified tackling the problem. Examples of these options are, type of speech (isolated words or continuous speech), number of speakers (single speaker system, multiple designated speakers, or unlimited population), type of speakers (male, female, child, cooperative, casual), speaking environment (booth, room, public place), transmission system (high quality microphone or telephone), type and amount of system training (no training, fixed training, continuous training), vocabulary size (small vocabulary (1 to 20 words), medium vocabulary, large vocabulary (more than 100 words)).

It can be seen from the above list that a wide variety of options and alternatives are available in the specification of a speech recognition system.

The block diagram of a speech recognition system depends much on the type of system that one wants to develop. For example the speaker independent, isolated digit recognition system differs very much from the large vocabulary, speaker dependent, isolated word recognition system. The various types of speech processing that is performed on the speech samples to do the recognition are: zero crossing rate, energy, LPC analysis LPC residual error, first autocorrelation coefficients, LPC log error etc.

One very fundamental unit of a speech recognition would be pole–zero classification. Same words spoken by different speakers have the poles and zeros quite close. Thus pole–zero classification can be used for speech recognition. Our implementation finds the poles and zeros from the speech samples. Using speech samples for words pole and zero tables can be prepared and then classified. These tables can then be used for speech recognition.

5.3 POLE–ZERO ANALYSIS

The usual pole–zero analysis techniques for determination of zeros are sensitive to the presence of fine structure in the spectrum. The reason for this is that, any technique for determination of the spectral zeros will be sensitive to the fact that between pitch harmonics the spectral amplitude approaches zero many times. This effect that may be misinterpreted as being due to the presence of spectral zeros. Thus, with any of the potential techniques for the measurement of spectral zeros, it is important to precede such an analysis with a deconvolution of speech.

In general there are three methods for such a deconvolution. One is by simply extracting individual pitch periods of speech waveform, and considering a single pitch period to correspond to the impulse response of the vocal tract. This would only be exactly correct if the vocal tract impulse response were shorter in duration than the pitch period. Techniques which rely on this method are generally referred to as pitch synchronous analysis. A second technique for implementation is based on the use of an all pole linear prediction analysis. If speech spectrum contains both poles and zeros, a very high order all pole analysis is required, since zeros must be approximated by poles. This high order all pole analysis then provides an approximation to the vocal tract impulse response or equivalently, the spectral envelope to which a lower order pole zero analysis can be applied. A third technique for implementing the deconvolution is through the use of homomorphic deconvolution to obtain an approximation to the vocal tract impulse response, to which a pole zero analysis can be applied.

A variety of pole–zero modeling techniques has been formulated theoretically; methods involving large matrix operations and iterative optimization techniques are generally unsatisfactory in the environment of speech analysis, in which memory and speed requirements are important and the potential for real time processing is often desirable. Thus the most suitable techniques tend to be those based on a least squares

error criterion, formulated in such a way as to involve the solution of linear equations. This generally leads to techniques in which the poles are estimated first, followed by an estimation of zeros, rather than a simultaneous estimation of both.

This is exactly what our algorithm will be doing in the implementation for the determination of poles and zeros.

5.4 THE ALGORITHM [12]

In the modified pole–zero decomposition technique that we are using, the cepstral coefficients are first calculated. The cepstral coefficients are necessary to calculate the group delays. The cepstral coefficients are computed from LPC coefficients by using a large model order and assuming the signal to be minimum phase. This sets all the roots of the model transfer function within the unit circle. The above assumption allows the use of properties of minimum phase polynomials. If the model transfer function A(z) is a 'M'th order polynomial in z^{-1} then:

$$\ln[A(z)] = \sum_{k=1}^{\infty} C_k z^{-1} \qquad (5.1)$$

where C_k's are the cepstral coefficients.

The above equation gives the relationship between the cepstral coefficients and the roots of the polynomial. If A(z) is considered to be an inverse filter model of the signal, then the filter coefficients are directly related to the cepstral coefficients. The following equations give the relationship:

$$C_1 = -a_1 \qquad (5.2)$$

$$jC_j = -ja_j - \sum_{k}^{j-1} kC_k a_{j-k} \quad \text{for } j=2,3,\ldots M \qquad (5.3)$$

$$jC_j = \sum_{k=1}^{M} (j-k)C_{j-k}a_k \quad \text{for } j=M+1,M+2,.. \qquad (5.4)$$

C_k's are cepstral coefficients, a_k's are LPC coefficients, M is the LPC model order

The minimum phase assumption also allows the use of the relationship

between the negative derivative of phase spectrum (group delay) and the cepstral coefficients. This helps in the evaluation of the group delays from the cepstral coefficients using the following equation

$$\Theta'(\omega) = \sum_{k=1}^{\infty} kC_k \cos(k\omega) \qquad (5.5)$$

where $\Theta'(\omega)$ is the negative derivative of the phase spectrum.

The positive and negative peaks in the negative derivative of the phase spectrum correspond to complex poles and zeros of the signal respectively. Convolution in the time domain is equivalent to addition in the cepstral domain. Separating the positive and negative group delay enables the computation of cepstral coefficients corresponding to poles and zeros and thus the decomposing of the signal into its pole part and zero part.

The cepstral coefficients of the original signal are thus split into pole and zero parts. Now we have separate cepstral coefficients for poles and zeros. These coefficients now allow the modeling of the pole part of the signal and zero part of the signal individually. Once again using relationships between cepstral and LPC coefficients (equations (5.2),(5.3) and (5.4)), the LPCs corresponding to the pole and zero parts of the signal are computed from the above cepstral coefficients that are already split to represent the pole and zero parts of the signal. Figure 5.1 describes the complete pole–zero decomposition procedure.

The final step that remains is to find the poles and zeros from the two separate polynomials we have. To generate a pole–zero table for words, the poles and zeros are calculated from the LPCs evaluated from the cepstral coefficients belonging to the pole and zero parts of the signal. The pole and zero portions are determined by simply factorizing the numerator and denominator LPC polynomials. If D(z) is the denominator polynomial represented by the LPCs that correspond to the pole part of the signal, then D(z) is given by:

$$D(z) = 1 - a_1 z^{-1} - a_2 z^{-2} .... - a_M z^{-M} \qquad (5.6)$$

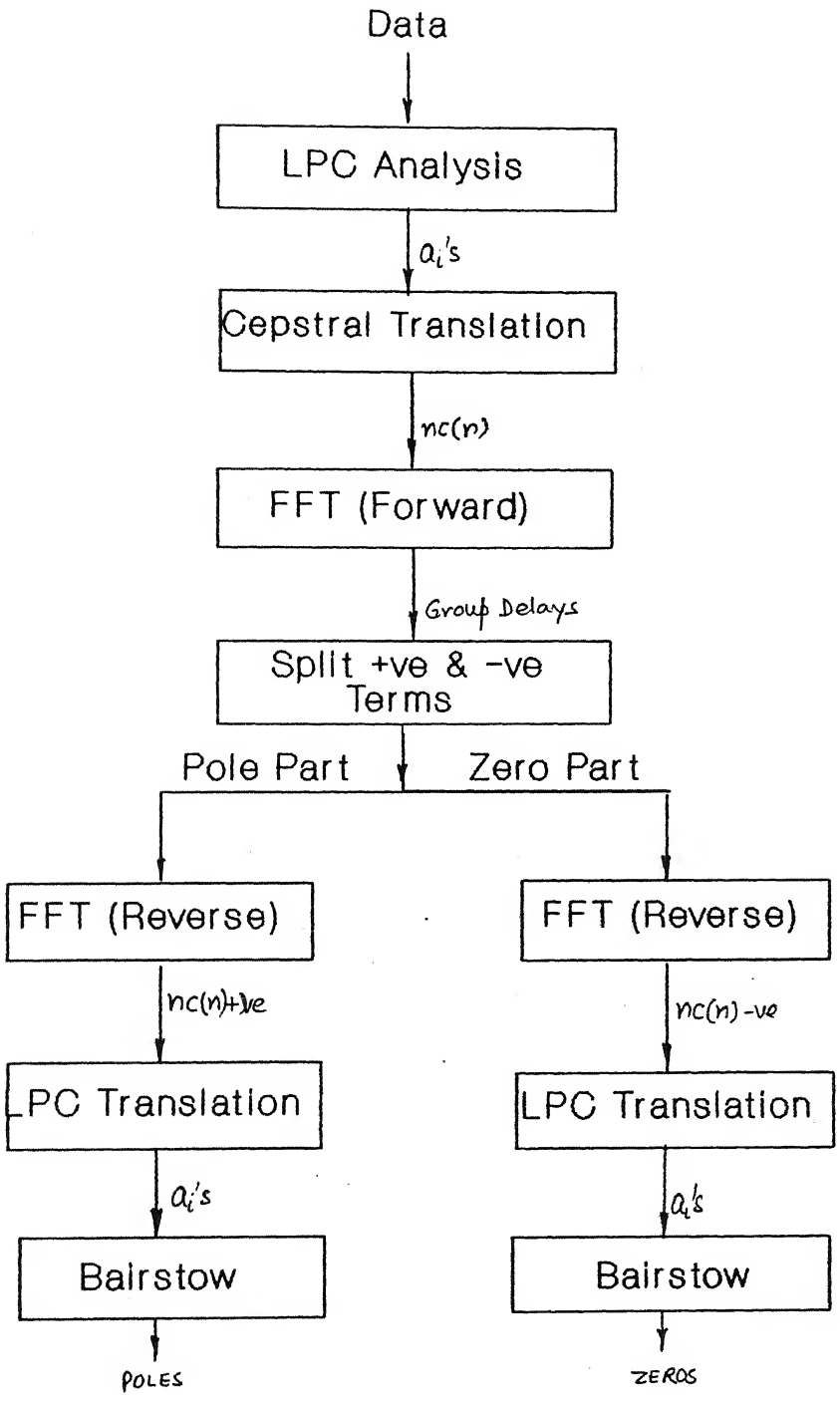


Figure 5.1 Flowgraph for calculating POLES and ZEROS

The roots of the polynomial would provide the 'pole' positions. Bairstow's method is used to determine the roots of the above polynomial by iteratively extracting quadratic factors and solving each factor for its roots.

$$Q(z)= (Z^2 + UZ + V) \qquad (5.7)$$

The roots p_1, p_2 are obtained from the first quadratic factor Q(z).

$$p_1,p_2 = \frac{-U \pm \sqrt{(U^2-4V)}}{2} \qquad (5.8)$$

The iteration process is repeated until all roots of the polynomial are extracted. The procedure is repeated for the numerator polynomial to determine the zeros.

5.5 IMPLEMENTATION [12,14]

Consider the Figure 5.1. The flow chart gives the whole pole–zero decomposition procedure. LPC routine is first of all used to calculate the LPC coefficients a_i's from the speech samples. From these LPC coefficients the cepstral coefficients C_i's are calculated using (5.2), (5.3), and (5.4). These cepstral coefficients are used as input to an FFT module which outputs the group delays. The FFT module is based on the Radix 2 decimation in time algorithm. The positive and the negative parts are split to give the pole and zero parts separately. Inverse FFT is used to get back the separated cepstral coefficients. LPC is used to get the separate polynomial coefficients for poles and zeros. Using Bairstow's method the polynomials are broken down in to quadratic factors which give the poles and zeros.

SUMMARY

In this chapter we have introduced speech recognition, and how pole zero analysis helps in the purpose. We have discussed pole–zero analysis and the algorithm we used for its implementation. The implementation of the algorithm has also been discussed.

CHAPTER SIX

FORMANT BASED PATTERN MATCHING OF VOICE

6.1 INTRODUCTION

In this chapter we discuss the basics of speaker recognition. Pattern matching is the first step to speaker recognition. We will see how pattern matching helps in speaker recognition. We have implemented a pattern matching algorithm. We will discuss the algorithm and our implementation for the algorithm.

6.2 SPEAKER RECOGNITION [4,5,15,16]

The first step towards speaker recognition in speech processing, is the use of pattern matching or pattern recognition techniques. This can be seen from the Figure 6.1. As we see in the Figure, a representation (pattern vector) of speech signal, spoken by a speaker, is obtained using certain digital speech processing techniques which preserve the features of the speech signal that are of use towards speaker identity. The resulting pattern is compared to previously prepared reference patterns and subsequent decision logic is used to make a choice among available alternatives.

The parameters that are selected for the purpose of speaker recognition should have certain qualities. The major features the parameter should have are : efficiency in representing speaker dependent information, easy to measure, should be stable over time, should be of natural occurance and its frequency of occurance in speech should be high, should change little with the environment, and as far as possible should not be suciptable to mimicry.

In the early works on speaker recognition spectroscopic data (time,

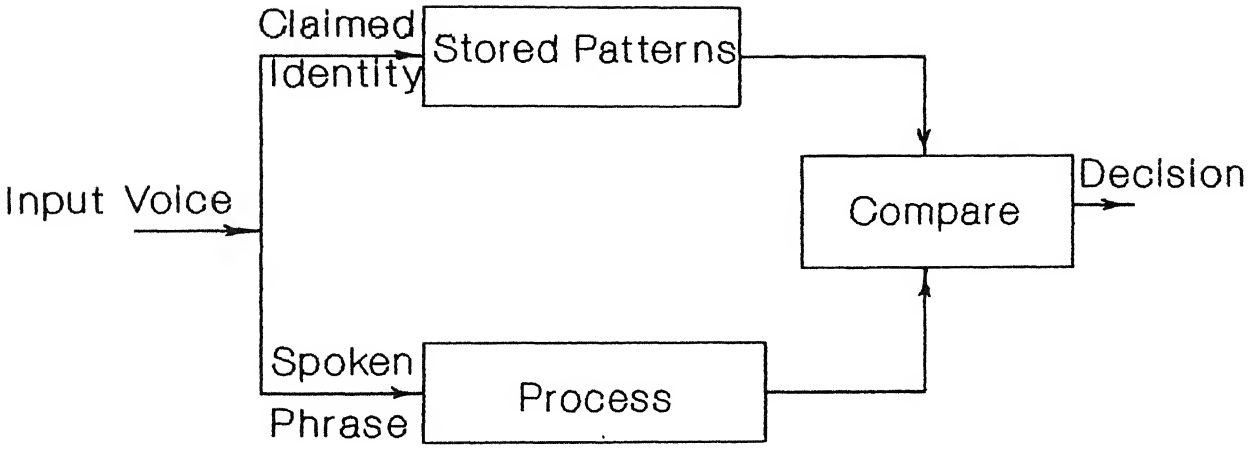


Figure 6.1 Use of Pattern Matching in Speaker Recognition

frequency, and energy pattern of speech) was used almost exclusively. The parameters are related to some property of short term power spectrum. In present day systems the parameters usually used are : Intensity, Pitch, Short time spectrum, Predictor coefficients, Formant frequency and bandwidth, Nasal coarticulation, Spectral correlation, and Timing and speech rate.

We would like to differentiate here that there are two distinct sub areas of speaker recognition. They are speaker verification and speaker identification. For speaker verification an identity is claimed by the user, and the decision required of the verification is binary i.e either the claimed identity is rejected or accepted. In order to make this decision, as we have indicated above, a set of features designed to retain essential information about speaker identity are obtained from one or more utterances of the speaker. The resulting measurements are compared (often using some highly nonlinear method of comparison) to a set of stored references patterns of the speaker the person claims to be. Thus for speaker verification only a single comparison between the set (or sets) of measurements, and the reference pattern is required to make the final decision to accept or reject the claimed identity. Generally a distance measure between the given measurements and the stored reference distribution is computed and an appropriate distance threshold is set for this purpose.

The problem of speaker identification differs significantly from the speaker verification problem. In this case the system is required to make an absolute identification of the speaker from among the a finite number of speakers in the user population.

We see from the above that there are a lot of similarities, as well as differences, between speaker verification and speaker identification systems. Each process requires a speaker to utter one or more test phrases, make some measurements on these test phrases, and then compute one (or more) distance functions between the measurement vector and the stored vector. Thus, in terms of the signal processing aspects of these two

problems, the methods are quite similar. The major differences occur in the decision logic. In one case only a comparison has to be made whereas in the other, a comaprison between a number of of speakers has to be made and a final result reached.

We now know that the signal processing aspects for the two types of systems are the same. We want to know what are the possible methods to achieve this recognition. Figure 6.2 shows the signal processing aspects of a speaker verification system. The only difference between this system and the speaker identification system is in the final distance measurement box where instead of making one distance measurement N distance measurements have to be made. After all the desired measurements have been made, the most important function is to compare the two patterns. It is not easy to compare the various temporal patterns such as pitch, intensity, formants because no speaker is able to speak at precisely the same rate for different repetitions of the verification phrase. The traditional solution to this difficulty has been to nonlinearly time warp the time scale of the input patterns to obtain the best possible registration between the stored reference patterns and the claimed speaker, and the measured patterns for the person's sample utterance. The basic concept of time warping is to warp the time scale 't' of a reference utterance so that significant events in some measurement contour line up with the same events in the reference contour. After the warping has been done the error at is added up at each sample to give the final distance measure.

The pattern matching discussed above is done on the time axis. Time axis pattern matching has been much in vogue, but pattern matching on the frequency axis is a much better option as the basic formants of an individual will give a better match and the error finally will be much less. For this reason we have used an algorithm for the pattern matching on the frequency axis. The algorithm is first described and then its implementation.

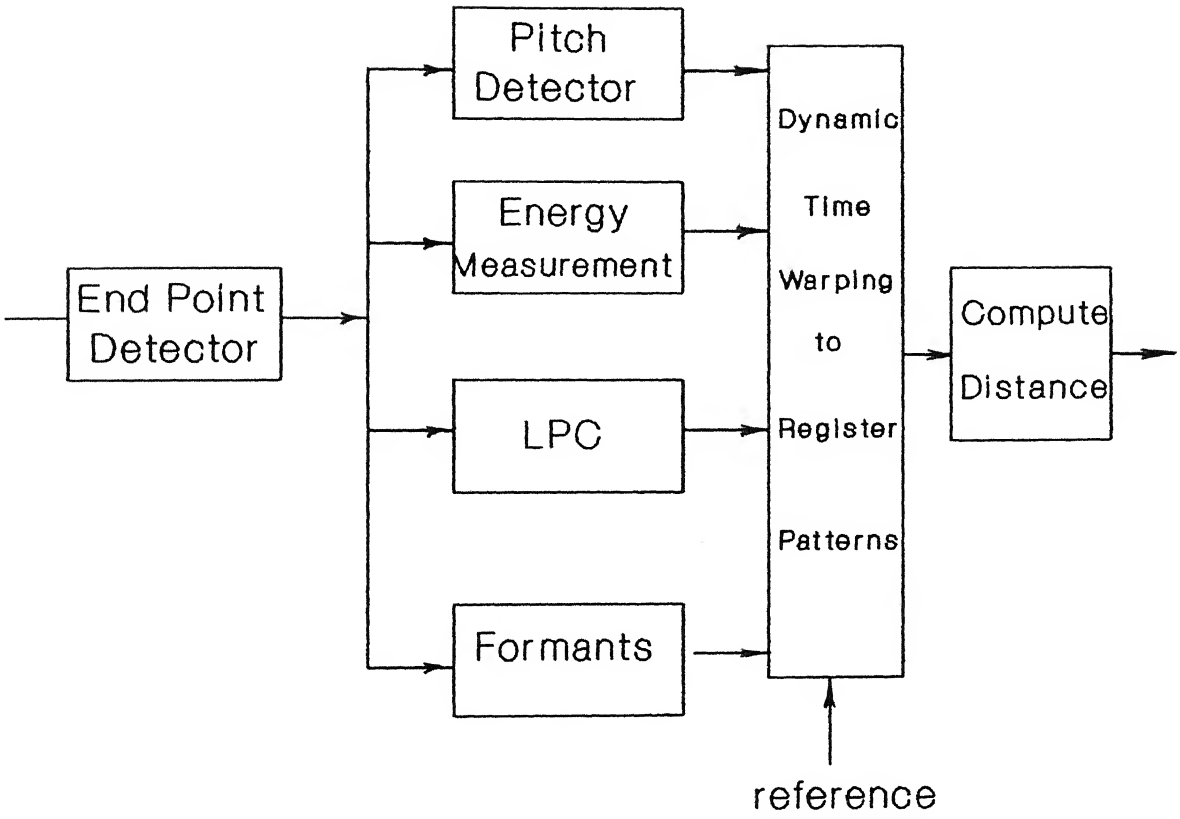


Figure 6.2 Signal Processing Aspects of a Speaker Recognition System

6.3 THE ALGORITHM [13]

In this section we discuss an algorithm on voice analysis and voice pattern matching. In the analysis, the complex conjugate and single real roots are obtained corresponding to the formants and gradients of voice spectrum respectively. In the algorithm the patterns are matched on the frequency axis for the distance between the input and reference voices which are expressed by the roots we have mentioned above. The roots of input voice are perturbed through cascade allpass filters which are adaptively controlled. The circuit parameter in each allpass filter section which corresponds to each formant frequency is controlled independently.

The pattern matching method is used to identify the voice pattern, keeping in mind that the above variations occur. The Dynamic Processing Matching (DPM) method had been used for pattern matching on the time axis for long, as no effective method was available for the matching on the frequency axis. Researchers have achieved success in pattern matching on the frequency axis. In the following subsections the analysis and pattern matching methods regarding the formant of voice spectrum have been discussed.

6.3.1 Voice Analysis

The algorithm in this subsection consists of the voice analysis. In the analysis, the roots of the transfer function on the z plane are obtained. A transfer function corresponding to a voice can be expressed as per the equation below:

$$X(z)= \frac{P_I}{1 + \sum_{i=1}^{I} \gamma_i z^{-1}} \qquad (6.1)$$

The format corresponds to the conjugate complex roots. The roots other than the above are single real roots. The equation can be resolved into factors as follows:

$$X(z) = \frac{p_I}{\prod_{l=0}^{L}(1-a_l z^{-1}) \prod_{m=0}^{M}(1+f_m z^{-1}+g_m z^{-2})} \quad (6.2)$$

$$= \frac{p_I}{\prod_{l=0}^{L}(1-a_l z^{-1}) \prod_{m=0}^{M}(1-c_m z^{-1})(1-c_m^{*} z^{-1})} \quad (6.3)$$

where $a_0 = f_0 = g_0 = 0$, $I = L + 2M$, p_I is gain, I is order, f_m and g_m are real coefficients, a_l is a single root, and c_m and c_m^{*} are conjugate complex roots, L and M are order of cascaded first order and second order respectively. The order M corresponds to the number of formants.

First of all we determine γ_i of Equation 6.1 using Linear Prediction. The Equation 6.1 is then mathematically converted to Equation 6.2 using Bairstow's method. From this, the roots of the equation (6.3) i.e a_l , c_m and c_m^{*} are obtained.

6.3.2 Theory of matching

The matching is considered on frequency axis on the roots corresponding to the input and reference voices.

In the time axis the matching was achieved using warping. To realize the matching on frequency axis, a method is needed to reshape the spectrum of the input voice according to some specified rule. This is called frequency control. The transfer function of a bilinear allpass filter (delay filter) is given by the equation:

$$T(z) = \frac{z^{-1} - \beta}{1 - \beta z^{-1}} \quad (6.4)$$

In the Equation 6.3, the delay element z^{-1} gives only a constant delay. When the allpass filter in the above Equation 6.4 is used instead of the constant delay, it becomes possible to control the delay according to the parameter β. The following is obtained as the new transfer function:

$$X(z) = \frac{P_I}{\prod_{l=0}^{L}(1-a_l\frac{z^{-1}-\alpha_l}{1-z^{-1}\alpha_l})\prod_{m=0}^{M}(1-c_m\frac{z^{-1}-\beta_m}{1-\beta_m z^{-1}})(1-c_m^*\frac{z^{-1}-\beta_m}{1-\beta_m z^{-1}})}$$

$$= \frac{P_I \prod_{l=0}^{L}(1-\alpha_l z^{-1})}{\prod_{l=0}^{L}(1+a_l\alpha_l)(1-A_l(\alpha_l)z^{-1})} * \frac{\prod_{m=0}^{M}(1-\beta_m z^{-1})^2}{\prod_{m=0}^{M}(1+\beta_m c_m)(1+\beta_m c_m^*)(1-B_m(\beta_m)z^{-1})(1-B_m^*(\beta_m)z^{-1})} \quad (6.5)$$

where $A_l(\alpha_l) = \frac{\alpha_l + a_l}{1 + a_l\alpha_l}$; $B_m(\beta_m) = \frac{\beta_m + c_m}{1 + \beta_m c_m}$;

and $B_m^*(\beta_m) = \frac{\beta_m + c_m^*}{1 + \beta_m c_m^*}$

The logarithmic expression of the transfer function is used by considering the perceptual characteristic of voice.

The cepstrum x(n) is given by the inverse of z transform of the logarithm of the Equation 6.5.

$$\text{i.e. } x(n) = Z^{-1}\{\ln X(z)\} \quad (6.6)$$

$$x(n) = \ln P_I - \sum_{l=0}^{L}\ln(1+a_l\alpha_l) - \sum_{m=0}^{M}\{\ln(1+c_m\beta_m)+\ln(1+c_m^*\beta_m)\} \text{ for } n=0$$

$$= \frac{1}{n}\Big[\sum_{l=0}^{L}\{-\alpha_l^n + (\frac{\alpha_l + a_l}{1 + a_l\alpha_l})^n\} + \sum_{m=0}^{M}\{-2\beta_m^n + (\frac{\beta_m + c_m}{1 + c_m\beta_m})^n + (\frac{\beta_m + c_m^*}{1 + c_m^*\beta_m})^n\}\Big] \text{ for } n>0 \quad (6.7)$$

It is possible to control the spectrum gradient and the formant frequency with α_l and β_m respectively in the Equation 6.7 above especially, because β_m corresponds to the formant frequency F_m, each formant frequency F_m can be independently controlled.

On the other hand, when assuming that the β_m is common to all m, it gives the dependent control scheme.

Realization of the matching system on the frequency axis is considered using the results obtained above where the input and the reference cepstrums s(n) and r(n) normalized by power are given by the equations below:

$$s(n) = -\sum_{l=0}^{L} \ln(1+a_l\alpha_l) - \sum_{m=0}^{M} \{ \ln(1+c_m\beta_m) + \ln(1+c_m^*\beta_m) \} \quad \text{for } n=0$$

$$= \frac{1}{n} \Big[\sum_{l=0}^{L} \{ -\alpha_l^n + \Big(\frac{\alpha_l + a_l}{1 + a_l\alpha_l}\Big)^n \} + \sum_{m=0}^{M} \{ -2\beta_m^n + \Big(\frac{\beta_m + c_m}{1 + c_m\beta_m}\Big)^n + \Big(\frac{\beta_m + c_m^*}{1 + c_m^*\beta_m}\Big)^n \} \Big] \quad \text{for } n>0 \quad (6.8)$$

$$r(n) = -\sum_{l=0}^{L} \ln(1+b_l\alpha_c) - \sum_{m=0}^{M} \{ \ln(1+d_m\beta_c) + \ln(1+d_m^*\beta_c) \} \quad \text{for } n=0$$

$$= \frac{1}{n} \Big[\sum_{l=0}^{L} \{ -\alpha_c^n + \Big(\frac{\alpha_c + a_l}{1 + a_l\alpha_c}\Big)^n \} + \sum_{m=0}^{M} \{ -2\beta_c^n + \Big(\frac{\beta_c + d_m}{1 + d_m\beta_c}\Big)^n + \Big(\frac{\beta_c + d_m^*}{1 + d_m^*\beta_c}\Big)^n \} \Big] \quad \text{for } n>0 \quad (6.9)$$

e(n) expressing the difference between input and reference cepstrums is defined as follows:

$$e(n) = s(n) - r(n) \qquad (6.10)$$

The evaluation function Q is given by the equation below:

$$Q = \sum_{n=0}^{N} \{ e(n) \}^2 \qquad (6.11)$$

The reference frequency control parameters α_c and β_c are fixed, and the input parameters α_l and β_m are adaptively controlled so that the evaluation function Q gets be minimized. Here, defining $\bar{s}(n)$ as the convergent value of input s(n) after the error is minimized. The distance d between the adaptively controlled cepstrum $\bar{s}(n)$ and the

reference r(n) as the distance between the two parameters. Then d is shown as follows:

$$d = \sum_{n=0}^{N} \{ \bar{s}(n) - r(n) \}^2 \qquad (6.12)$$

6.4 IMPLEMENTATION

The implementation of the analysis part has been done first. The first module uses LPC to get the coefficients of Equation 6.1. Bairstows method is then used to factorize the equation as per Equation 6.2. A transform equation converts it to the form of Equation 6.3. Then the reference cepstrum is generated. The input cepstrum is also obtained. The final error is then minimized and the final distance measure obtained.

SUMMARY

In this chapter we discussed speaker recognition. We then discussed the algorithm for pattern matching which can implement speaker recognition.

CHAPTER SEVEN

RESULTS and CONCLUSION

7.1 INTRODUCTION

In this chapter we discuss the final results we have got from the tests on actual data. We have also implemented some other algorithms but they have not been tested fully. We also list these algorithms explaining in short how they have been implemented.

7.2 RESULTS

7.2.1 CODEC

In the CELP implementation, the speech samples used are sampled at 20 KHz. The size of the codebook used is: 30 vectors. The program is tested using two sample sizes 60 and 120. The order of the LP filter used is 10. The whole process of coding for the sample size of 60 takes of the order of 100000 cycles. At 125ns per cycle operation, this gives us a bit rate of 12800 bits/s for the particular processor with a register size of 16 bits. When using the CODEC with a sample size of 120 the operation takes approximately 128000 cycles. This gives a bit rate of 10000 bits/s.

Tests performed on the codec using five speech waveforms (four of which are vowels) show that in the first case when using the CODEC with sample size 60 the results are as follows. In the case of vowels the error is around 5% of the input voice sample in terms of magnitude. In the case when it is not a vowel the error is higher and can be as high as 10% of the input. In the second case i.e. with a sample size of 120 the results are as

follows. The error magnitude in all the cases compared with the results of the smaller sample size CODEC are higher. The error are of the order of 7.5% to 10% in case of vowels and still higher in the case the input is not a vowel.

A program for the CODEC on the PC XT using "C" takes of the order of 14 to 15 seconds for the same operations though the error is a bit lesser.

The input, output and the error between the input and output is shown in the form of graph plots. Plots of the outputs of the final results are attached. The set of plots in Figure 7.1.1 to 7.1.5 show the plots for the speech inputs. Figures 7.2.1 (a) & (b) to Figures 7.2.5 (a) & (b) show the output and the error between the input (shown in Figures 7.1.1 to 7.1.5) and output when the CODEC is used with sample size of 60. In the plots the magnitude for the speech samples is plotted along the Y axis and the X axis represents the time axis. When plotting the error the Y axis scale is magnified by an order of 50.

A specific test is not required for the codec as the speech is regenerated. It therefore acted as a self test. As we see that a bit rate of 12800 bits/s and 10000 bits/s is within real time operation our program is well suited for real time operations.

7.2.2 POLE–ZERO

The pole zero algorithm takes a very small amount of time for its operation. The whole operation for the implementation takes around 45000 cycles. The sizes of input and order of the filters used are of the same order as in the CELP implementation. One cycle of the processor takes 125ns. Thus pole zero generation takes approximately 5.6ms. The purpose of real time speech recognition can be well achieved at this rate. Plots showing the poles and zeros for the speech waveforms segments are attached. Figures 7.3.1 to 7.3.5 show the pole zero plots for the first set of speech samples (first 60 samples) of the speech waveforms in Figures 7.1.1 to 7.1.5.

The poles and zeros generated can be best tested by regenerating speech with the poles and zeros we have obtained. A program to generate speech back using these

poles and zeros is used as a test. In the synthetic speech generated the error between the input and output is very less indicating that poles and zeros generated are correct. A program for the same purpose with same size and data written in "C" and tested on the PC XT takes a very long time for operation (of the order of 20secs).

7.2.3 FORMANT BASED PATTERN MATCHING OF VOICE

The pattern matching algorithm takes of the order of 90000 cycles for its operation. This means approximately 11.25 ms. The program in "C" on the PC takes of the order of 25 to 30sec. All data sizes and order of filters used are of the same order as in the CELP algorithm. The rate of operation at 11.25 ms is possible for the operation in real time for the size of data used.

The program is tested using the speech inputs of Figure 7.1.1 to 7.1.5 as the stored waveform and the test waveforms are the same waveforms with a finite amount of noise added to it. The amount of error for the same inputs is zero but as the noise component is increased the error too increases. Testing is also done by giving the test samples some delay. The error increases. This shows the program to be correct. The error between the errors generated by the two methods (ADSP 2100 and PC) was 5% to 10%. Table 7.1 shows the distances between the speech waveform and another speech waveform.

7.3 ADDITIONAL ALGORITHMS IMPLEMENTED

Some algorithms are of great importance in speech processing applications. Some additional algorithms implemented are not been fully operational but are of use in speech processing applications. The algorithms are a sample and hold program for sampling speech spoken through a microphone and a program for the determination of pitch.

7.3.1 Sample and Hold

Sample and hold is not a simple affair as one might think. In the ADSP 2100 more care has to be taken for the purpose because the CODEC in the ADSP 2100 represents input and output in 8 bit binary form, and by using the μ–law nonlinear transformation the CODEC extends the dynamic range to an effective 13 bits. Thus when data is obtained from the CODEC it has to be linearized to ensure that linear signals are processed. We would like to mention here that the CODEC samples data at a frequency of 8.192 kHz using its own dedicated clock generator.

The sample and hold program samples the values and then it is linearized to ensure that the program is run on corrected data.

7.3.2 Pitch

We have already discussed pitch as a topic in the chapter on bandwidth compression techniques. In this case we use an algorithm in which after we have the LPC coefficients a_k's they are autocorrelated and these autocorrelated values of a_k's are crosscorrelated with the autocorrelated values of the original input data. The ratio of the maximum value of the corsscorrelated value to the first value of the sequence serves as a measure for the purpose of determining whether the windowed sequence is voiced or not. If it is voiced the peak value is multiplied by the sampling period to give the pitch. If it is unvoiced it returns a value zero.

7.4 SCOPE FOR FUTURE WORK

The work in the thesis, achieves three purposes which are a CODEC, a POLE–ZERO analysis and PATTERN MATCHING on the frequency axis. First of all the final data book for the pole–zero can be prepared or an implementation for it can be written. This data book can be used for the purpose of speech recognition. For this purpose

a decision logic has to be implemented. In the formant based pattern matching algorithm also a data book and a decision logic algorithm can be implemented for the purpose of speaker recognition.

In the field of speech processing a lot can be done to achieve a whole speech processing system. To know the starting of the speech signal we need an end point detector. An end point detector is a necessity for a real time system. Speech has also to be divided into frames for the operation of the systems. There are other areas of speech processing which have not been implemented, like speech synthesis and aids for the handicapped. Speech that will be available will always have a lot of noise and even some part might be missing, so algorithms can be implemented to overcome these difficulties.

SUMMARY

In this chapter we discussed the results we got in our implementation of the algorithms. It should be noted that the speech samples we used were sampled at 20 khz but the chip ADSP 2100 can sample only at a 8.192 khz. If we use the sampler on ADSP 2100 board we will get a much faster performance and a much lower bitrate but the overall error will be higher.

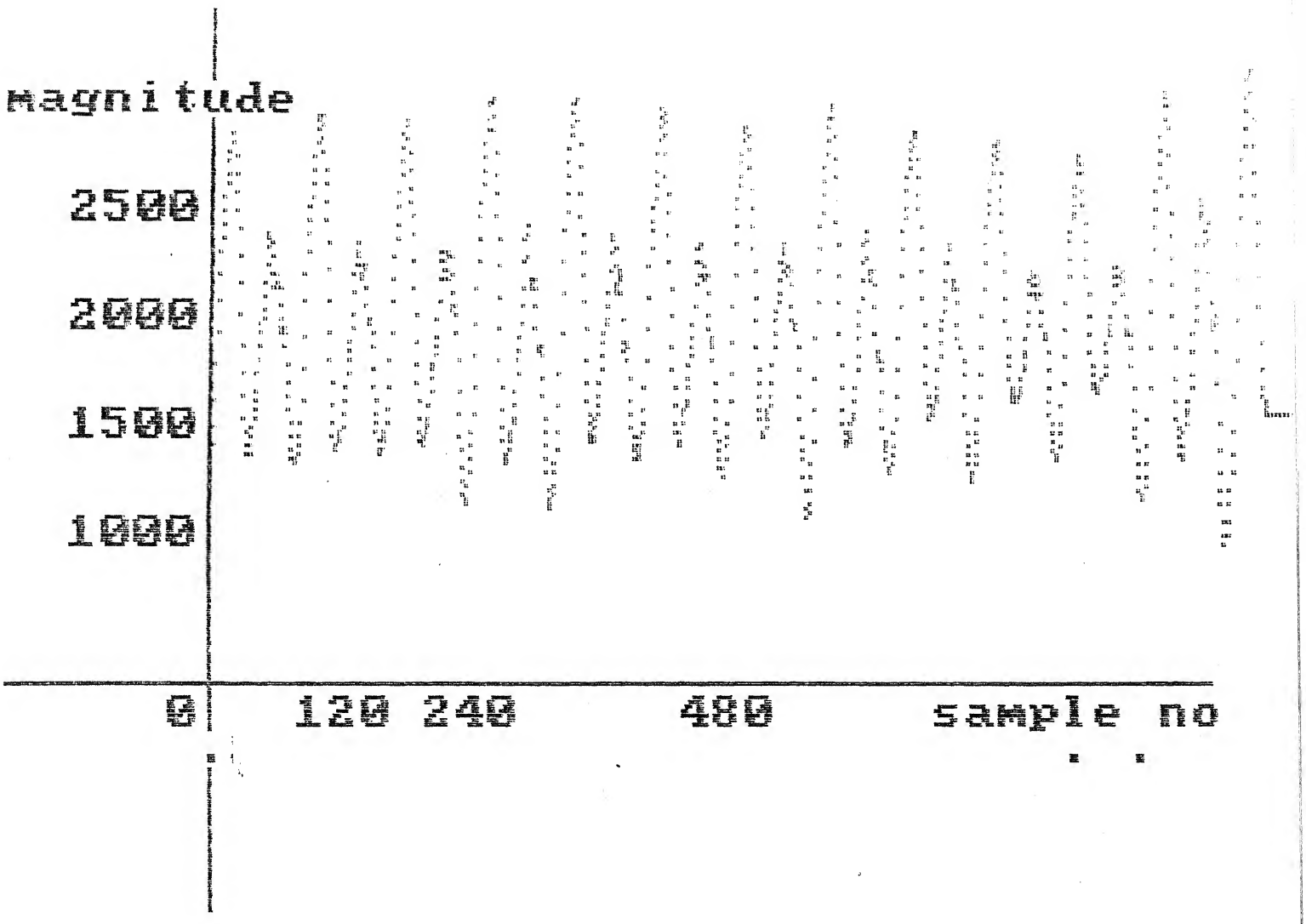


Figure 7.1.1

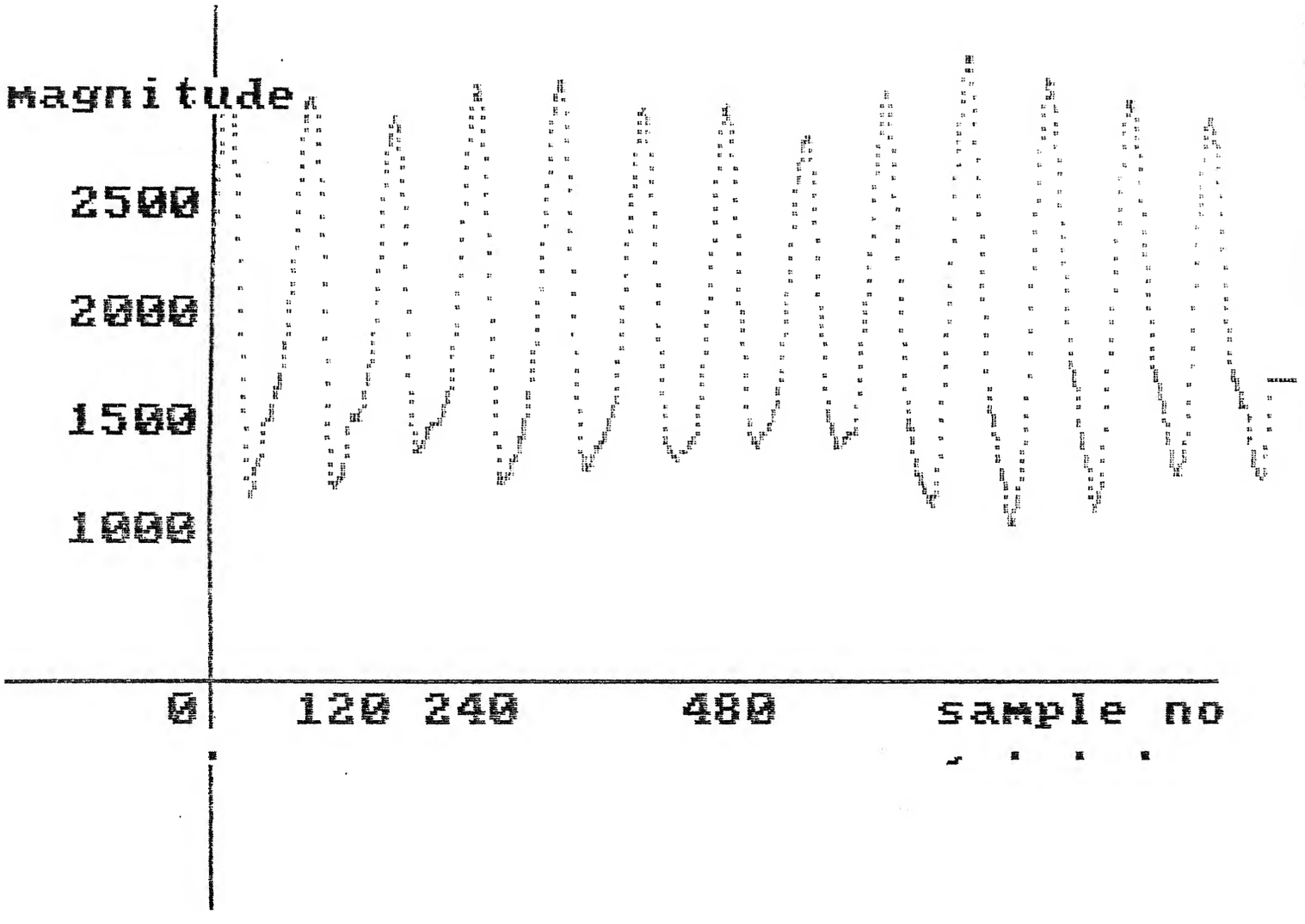


Figure 7.1.2

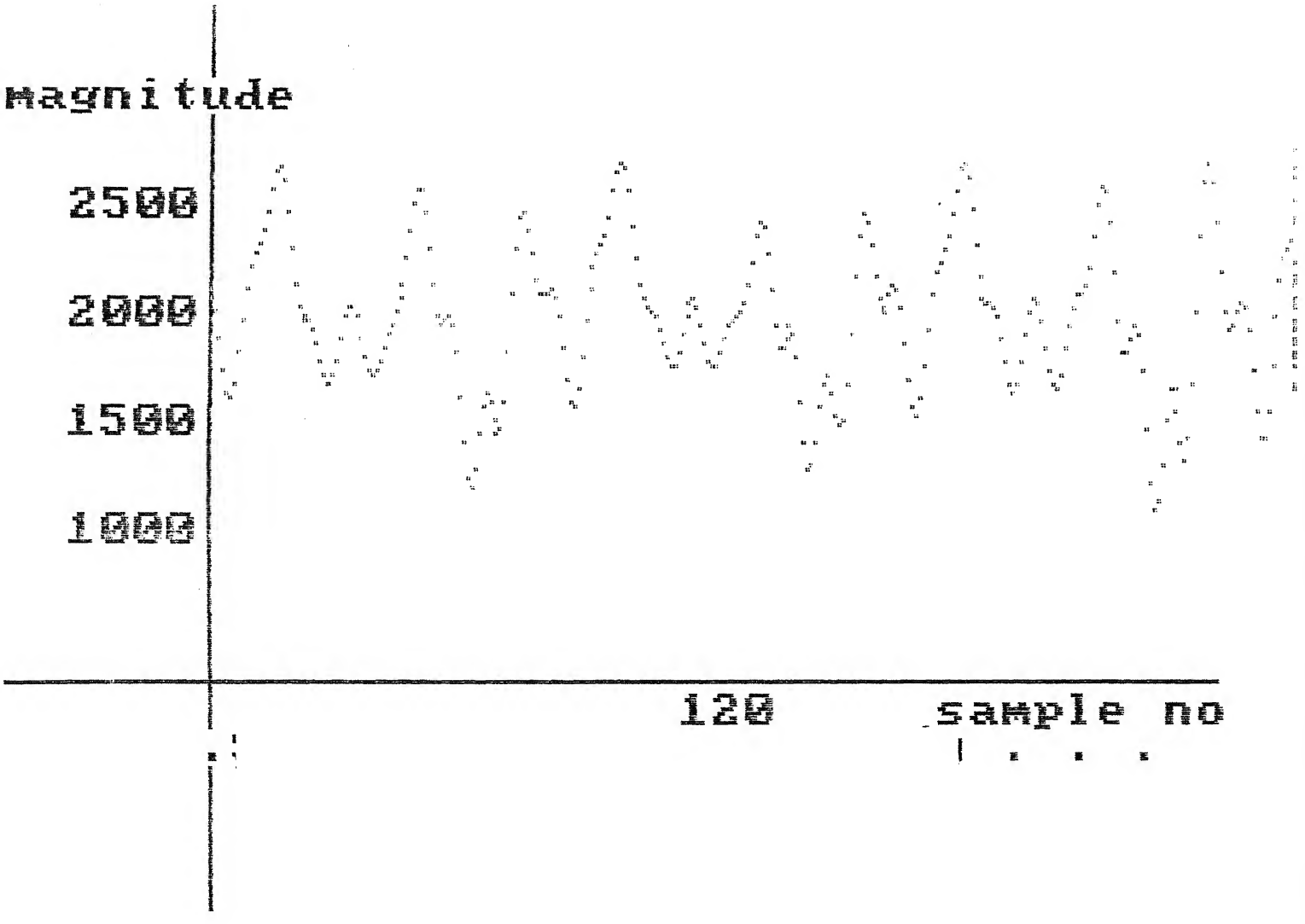


Figure 7.1.3

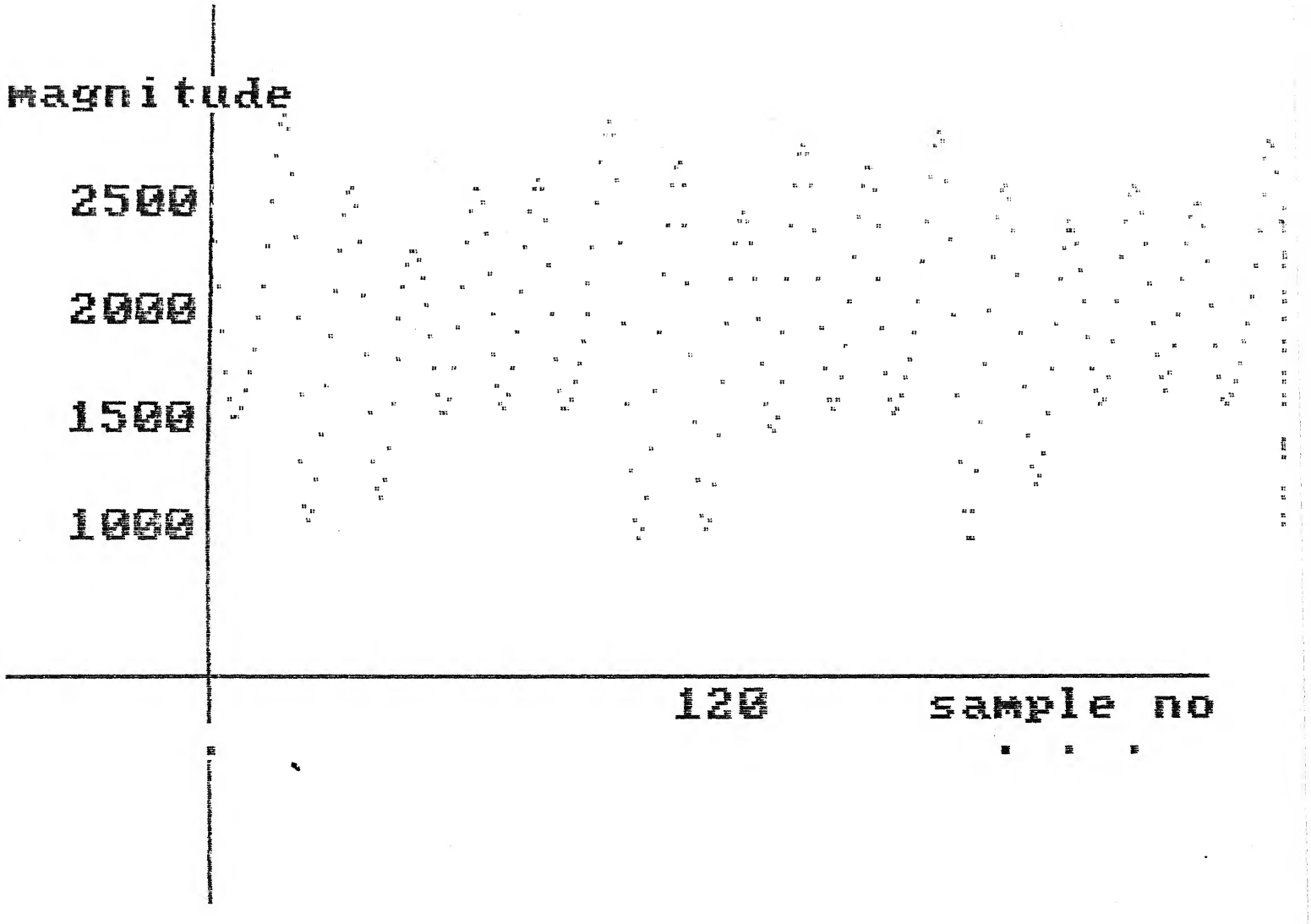


Figure 7.1.4

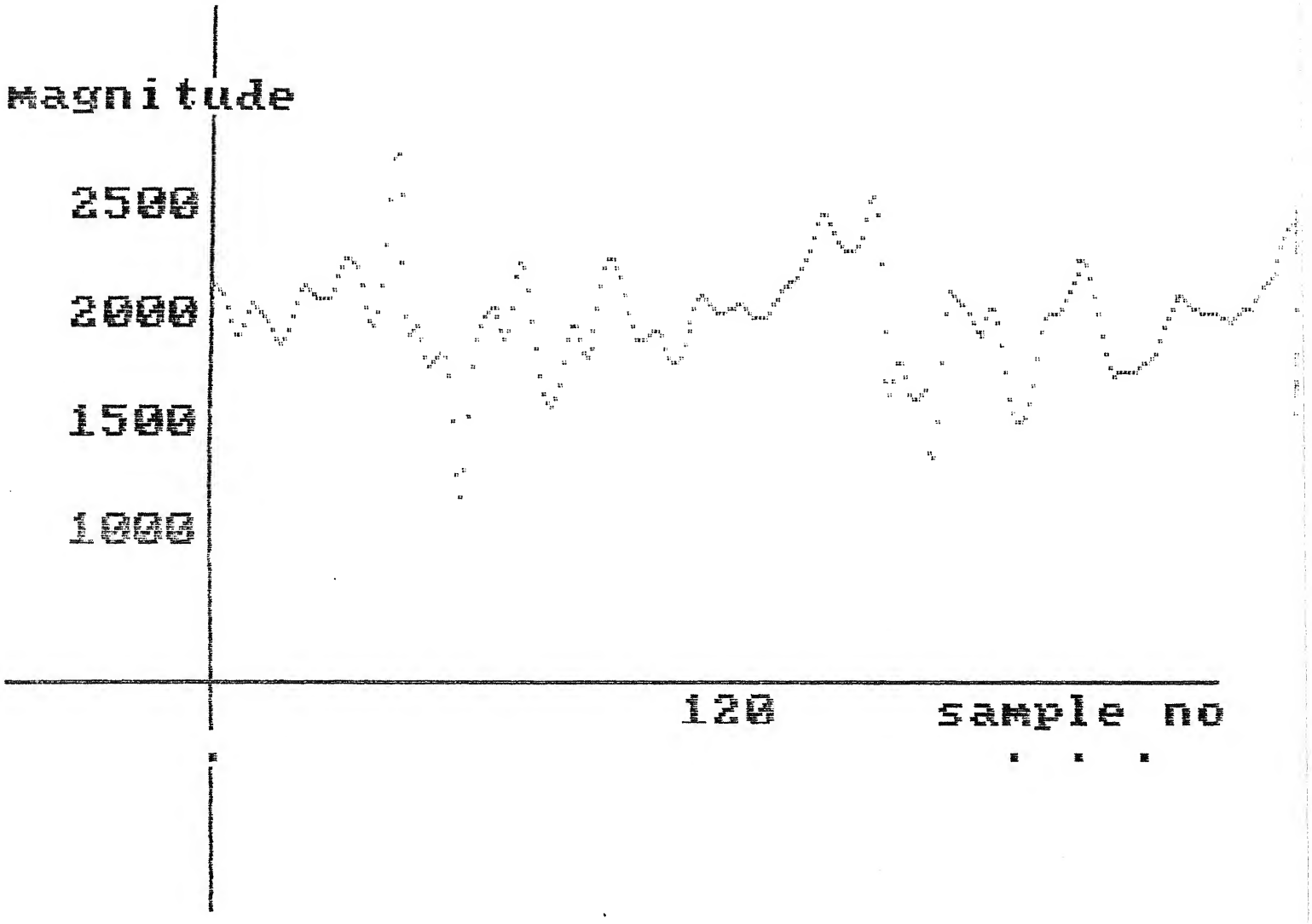


Figure 7.1.5

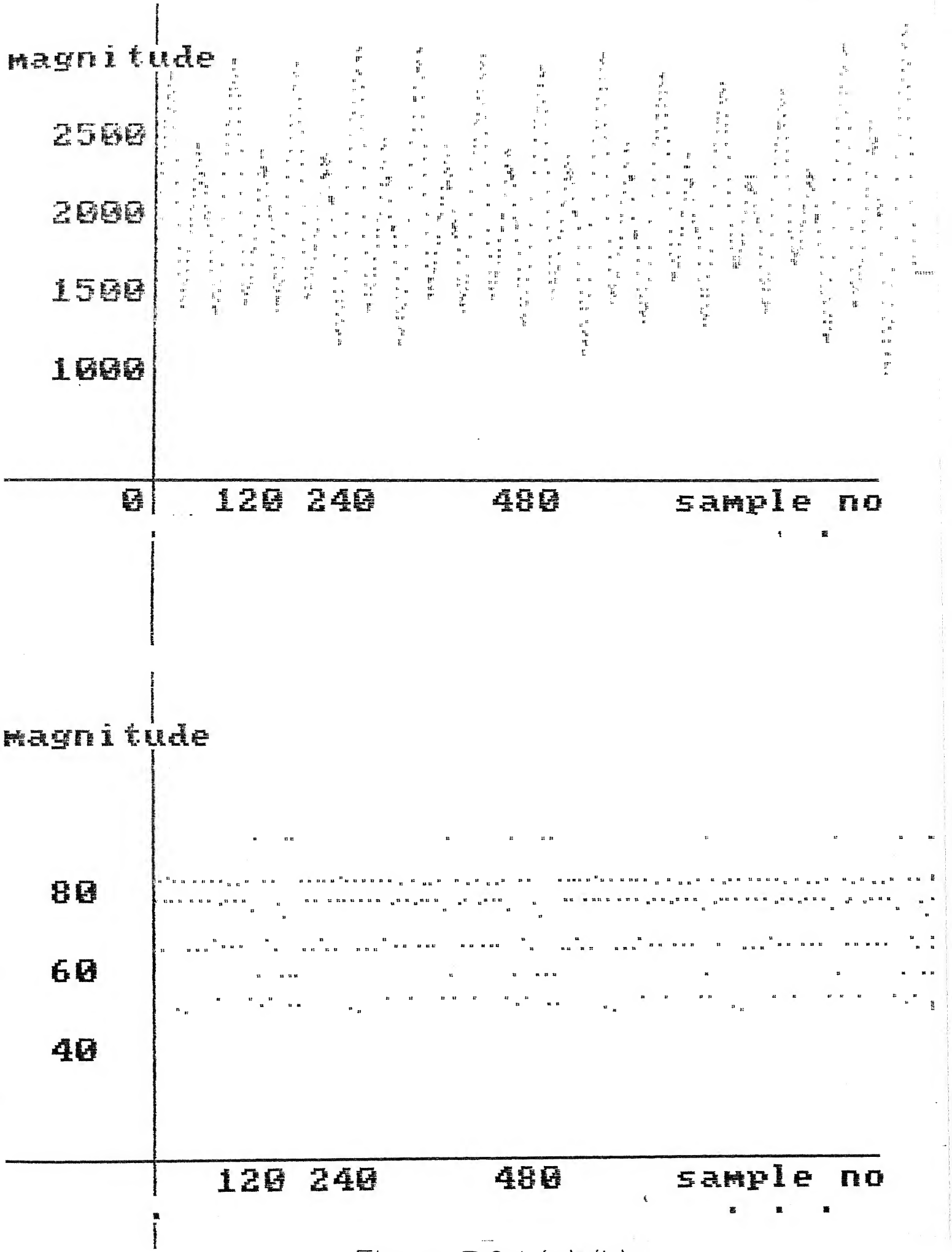
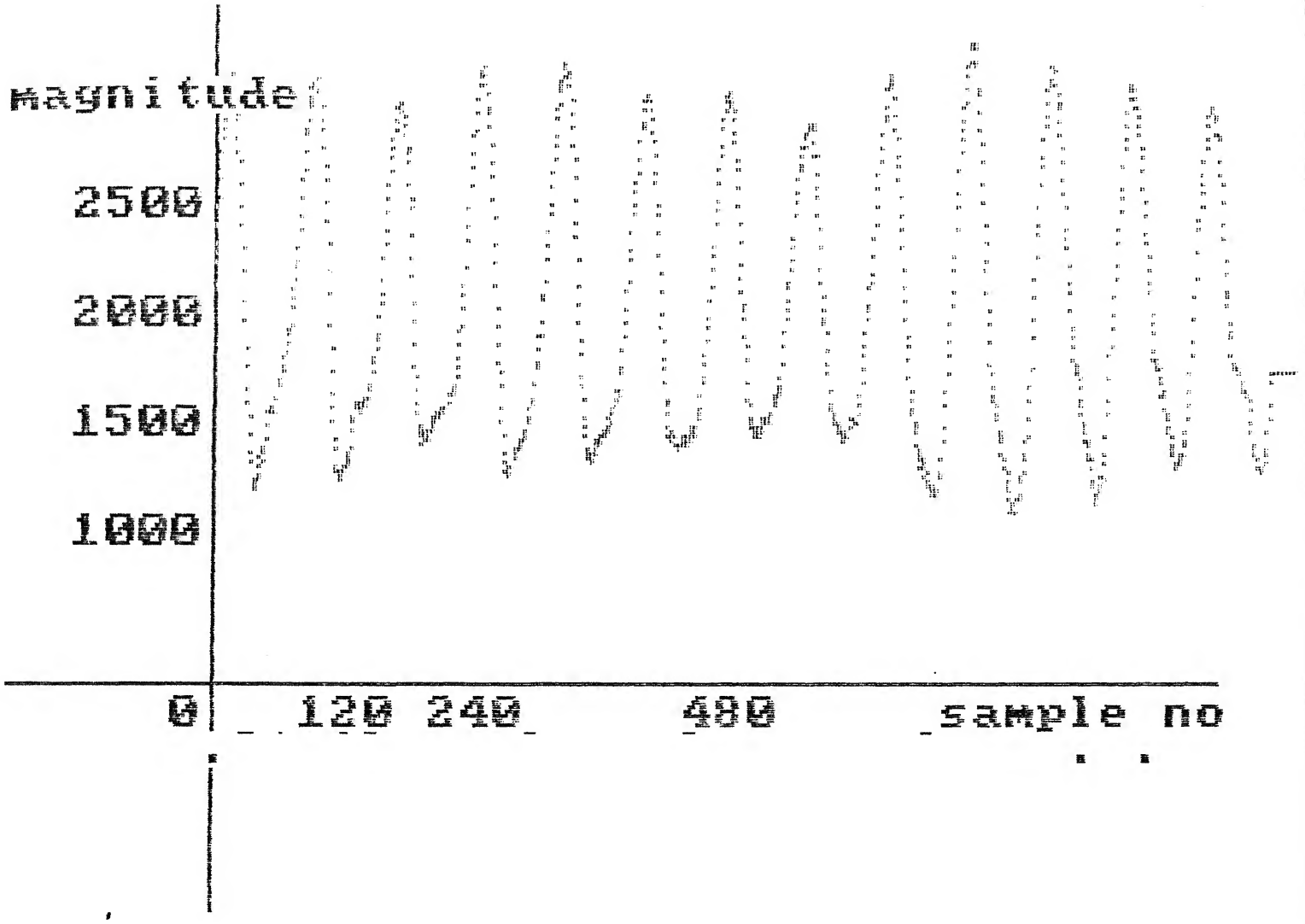
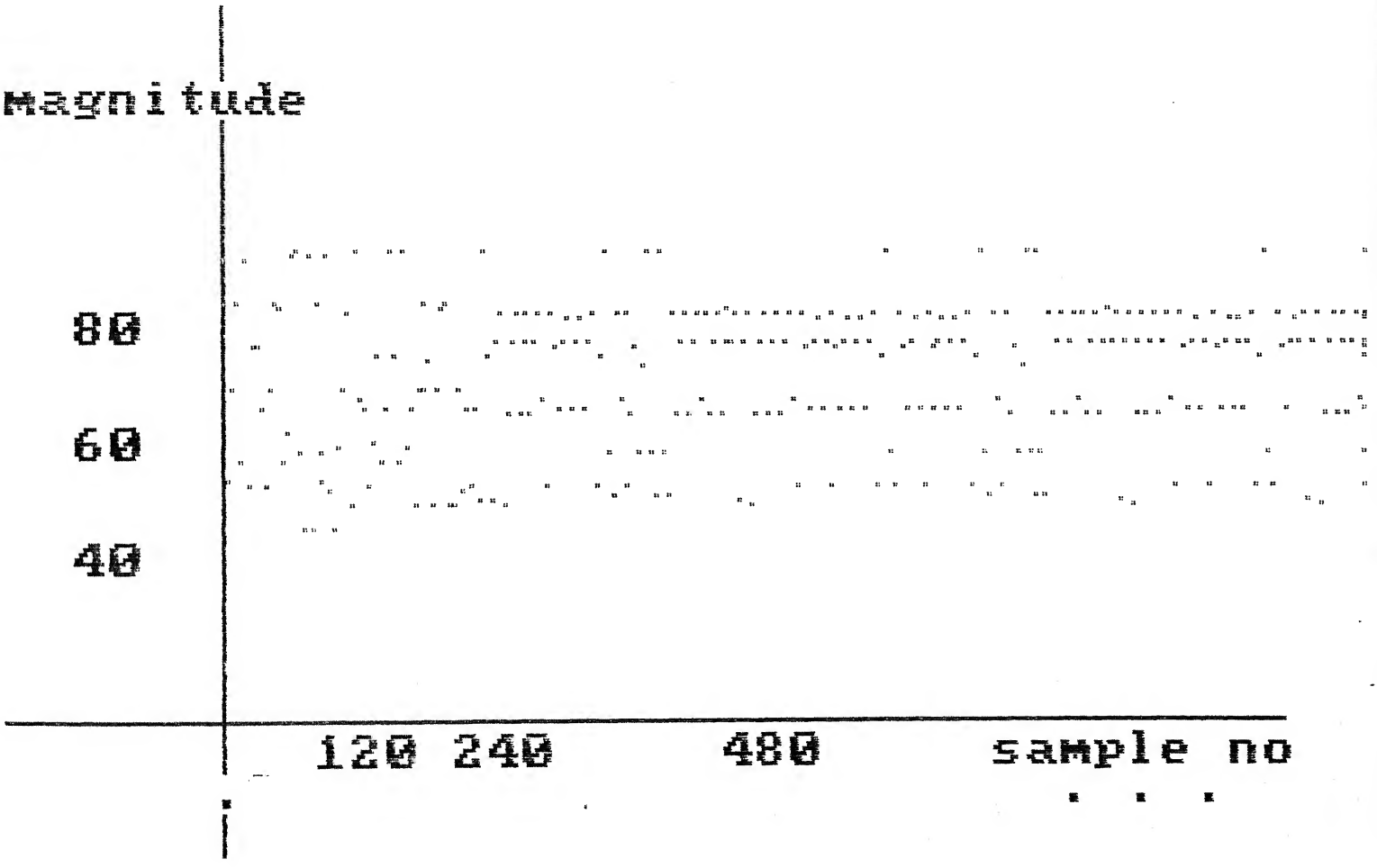
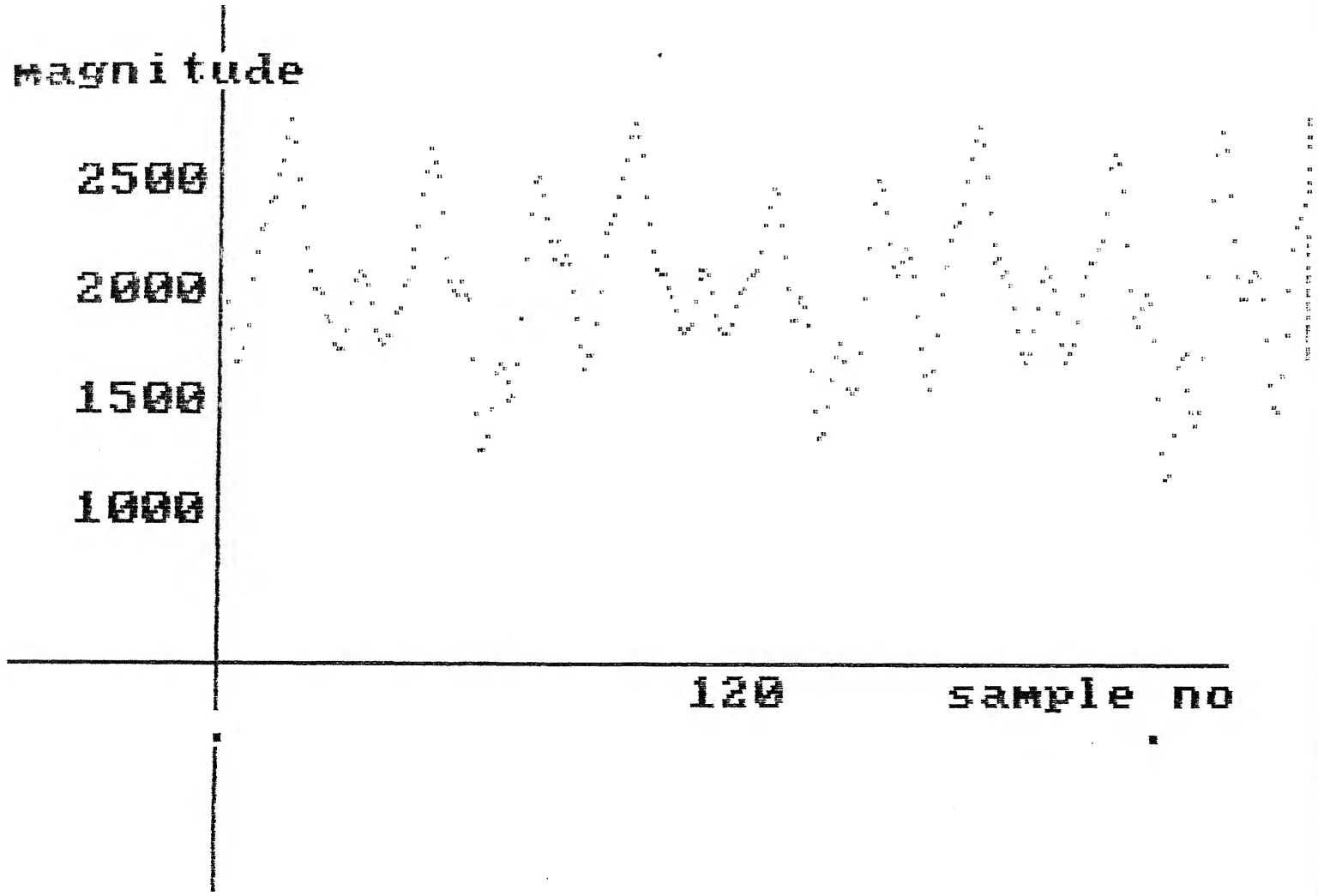


Figure 7.2.1 (a)&(b)


magnitude
2500
2000
1500
1000
0
120
240
480
sample no


magnitude
80
60
40
120
240
480
sample no



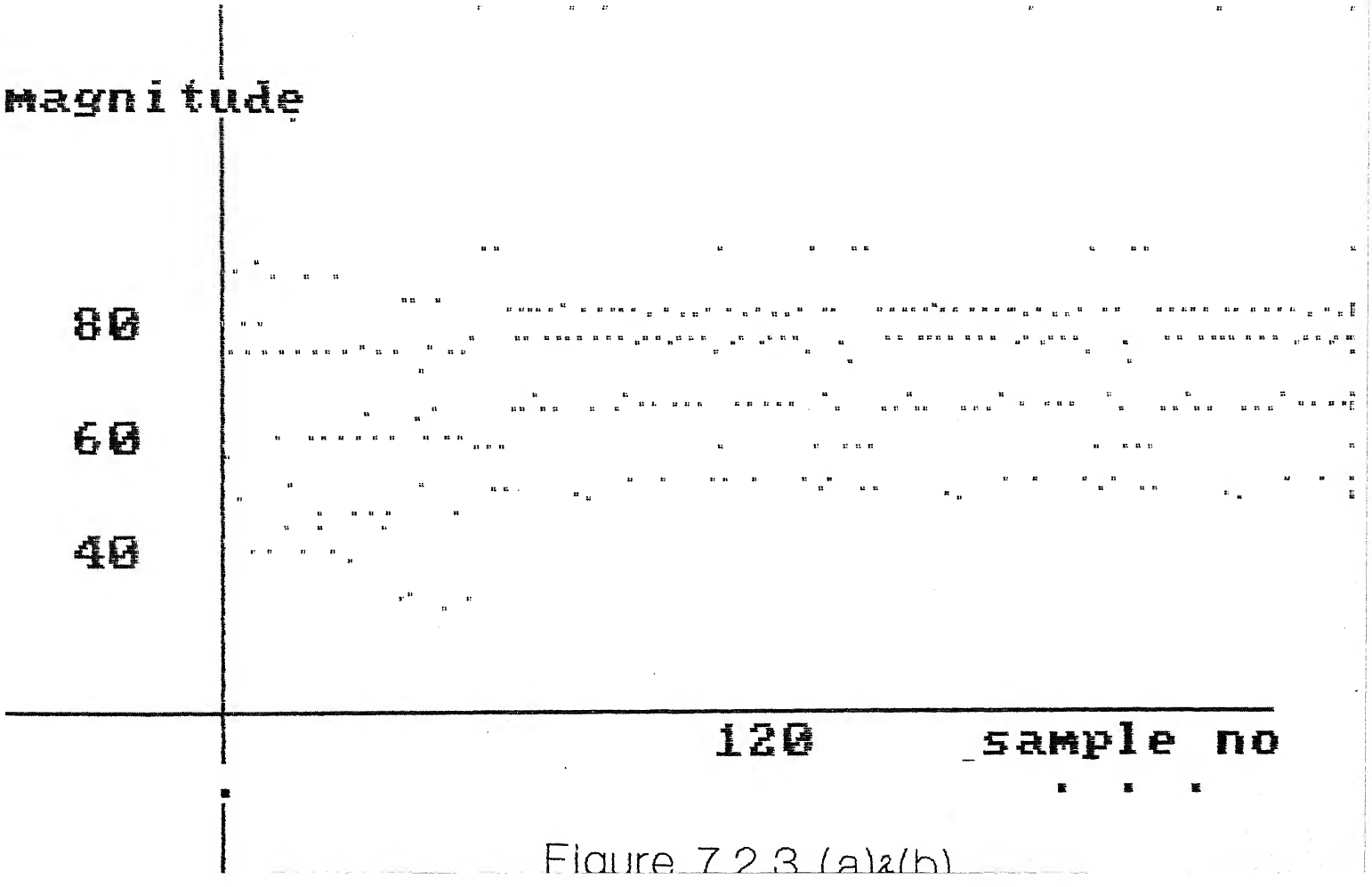


Figure 7.2.3 (a)&(b)

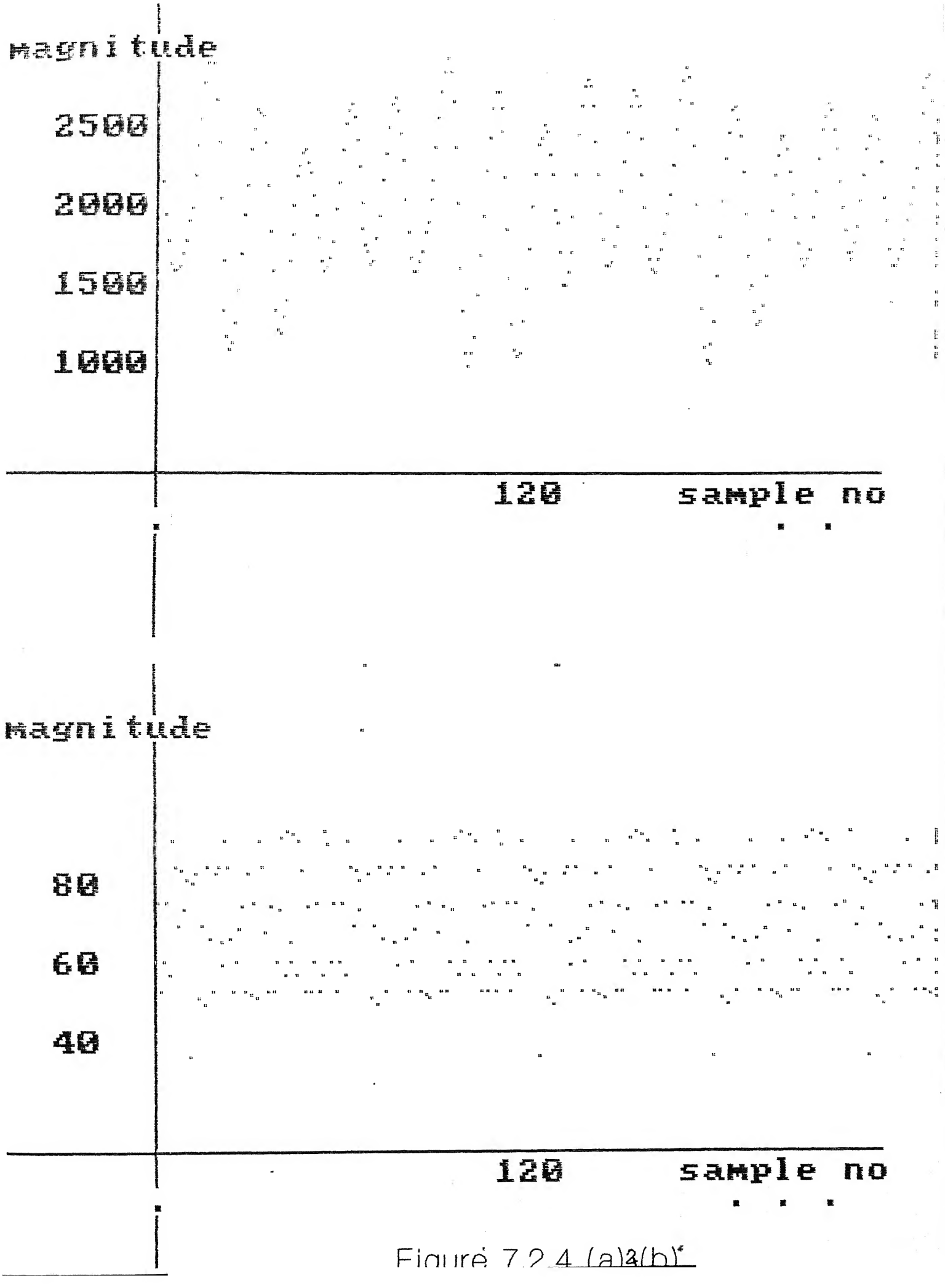


Figure 7.2.4 (a) & (b)

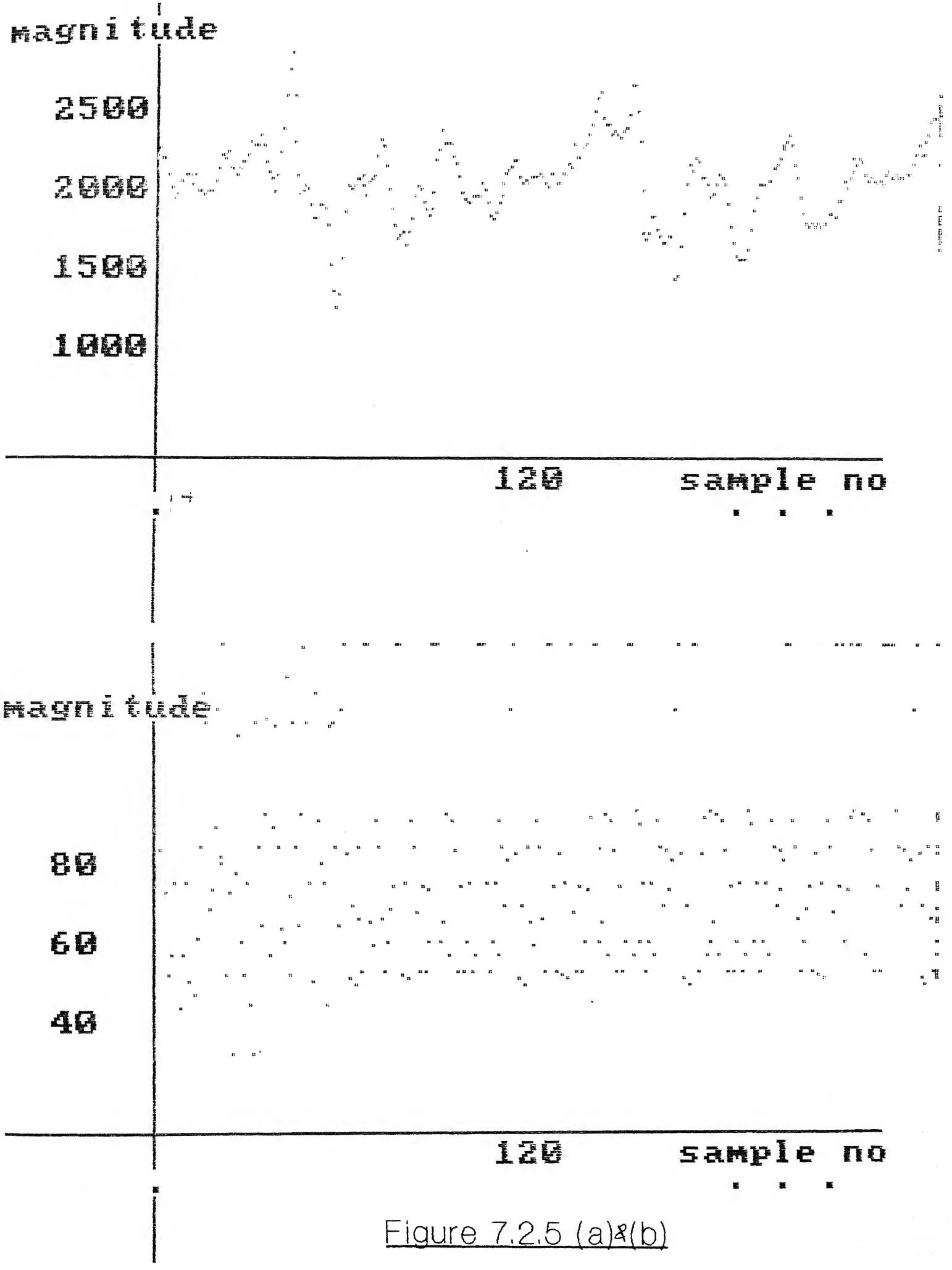


Figure 7.2.5 (a)&(b)

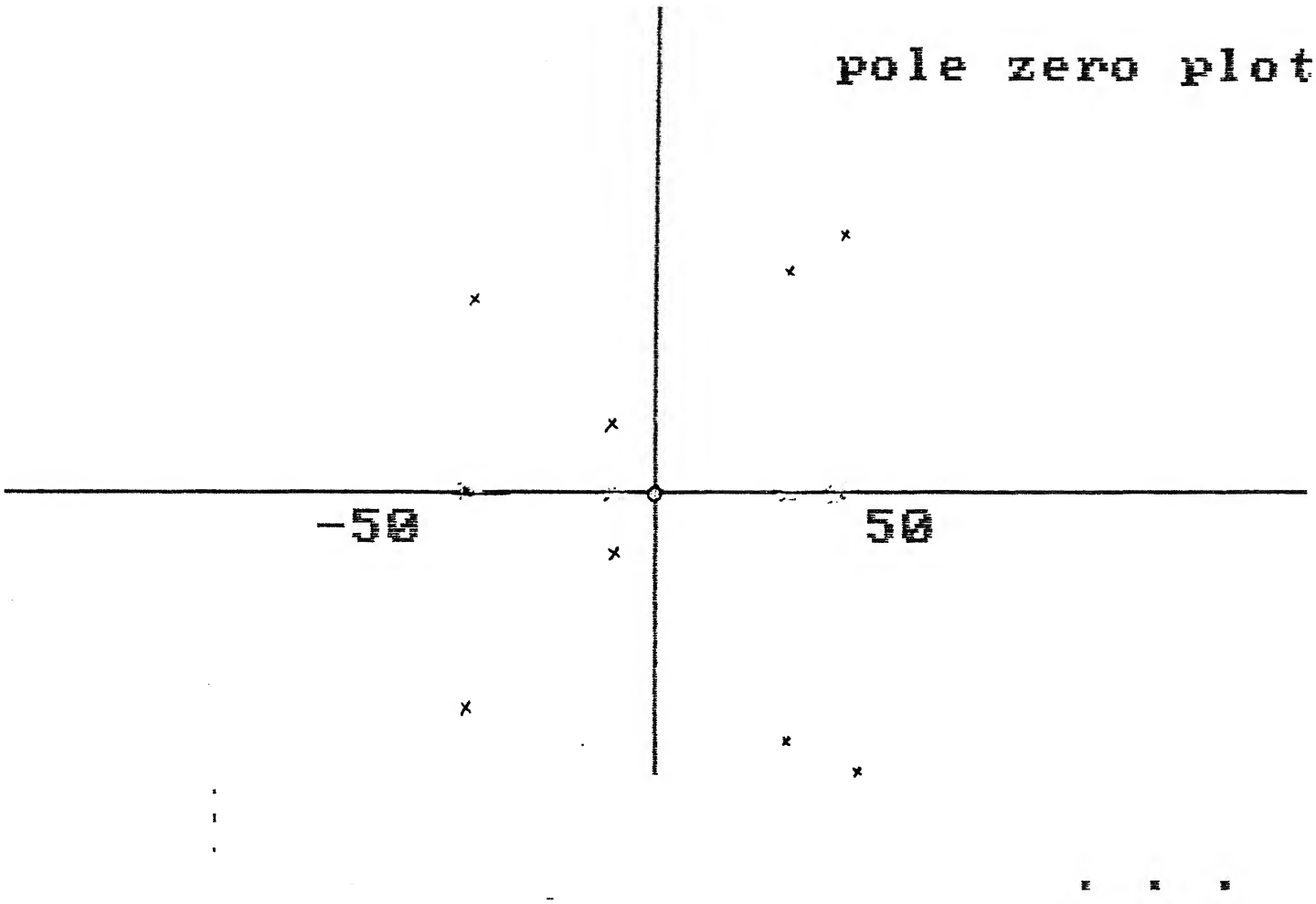


Figure 7.3.1

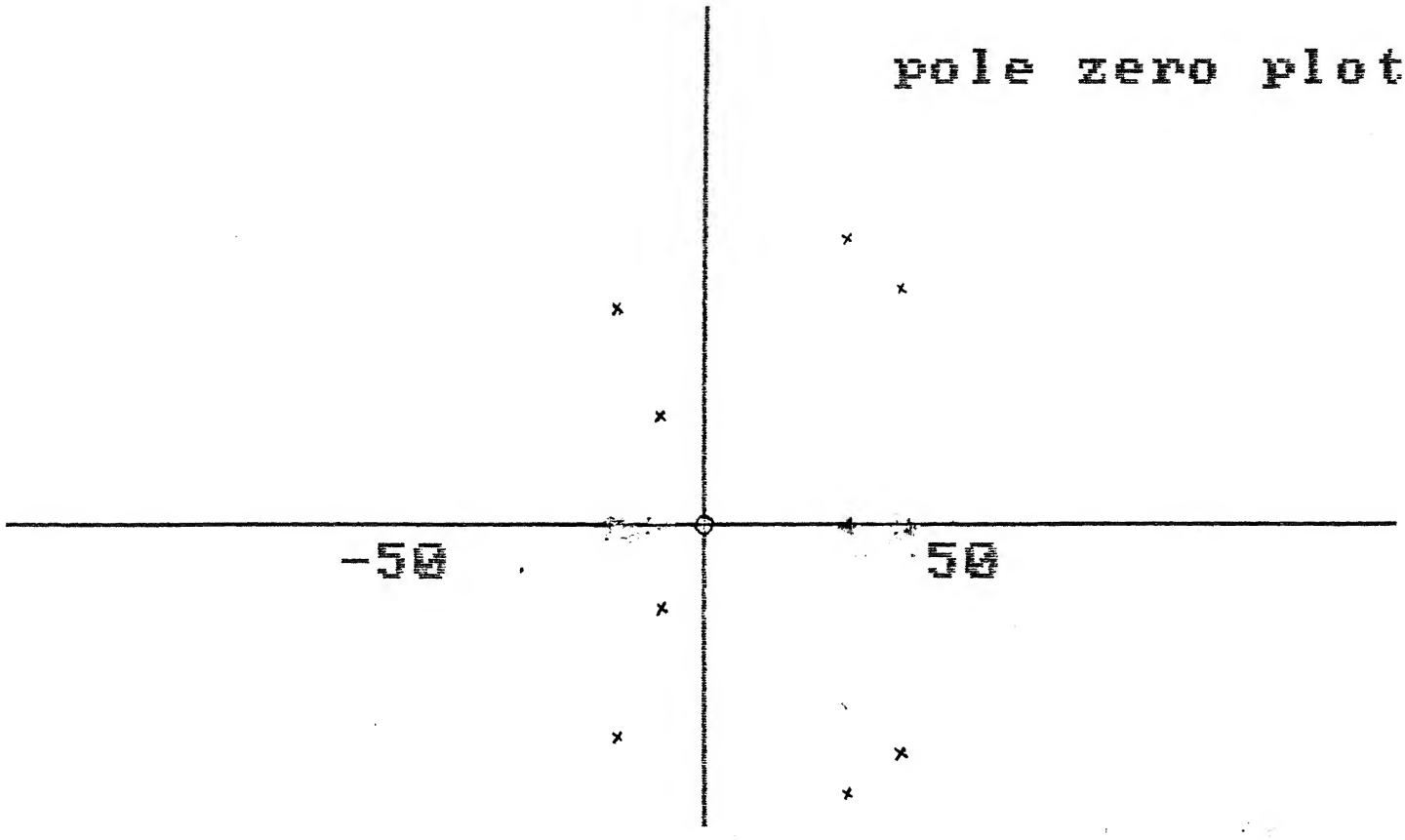


Figure 7.3.2

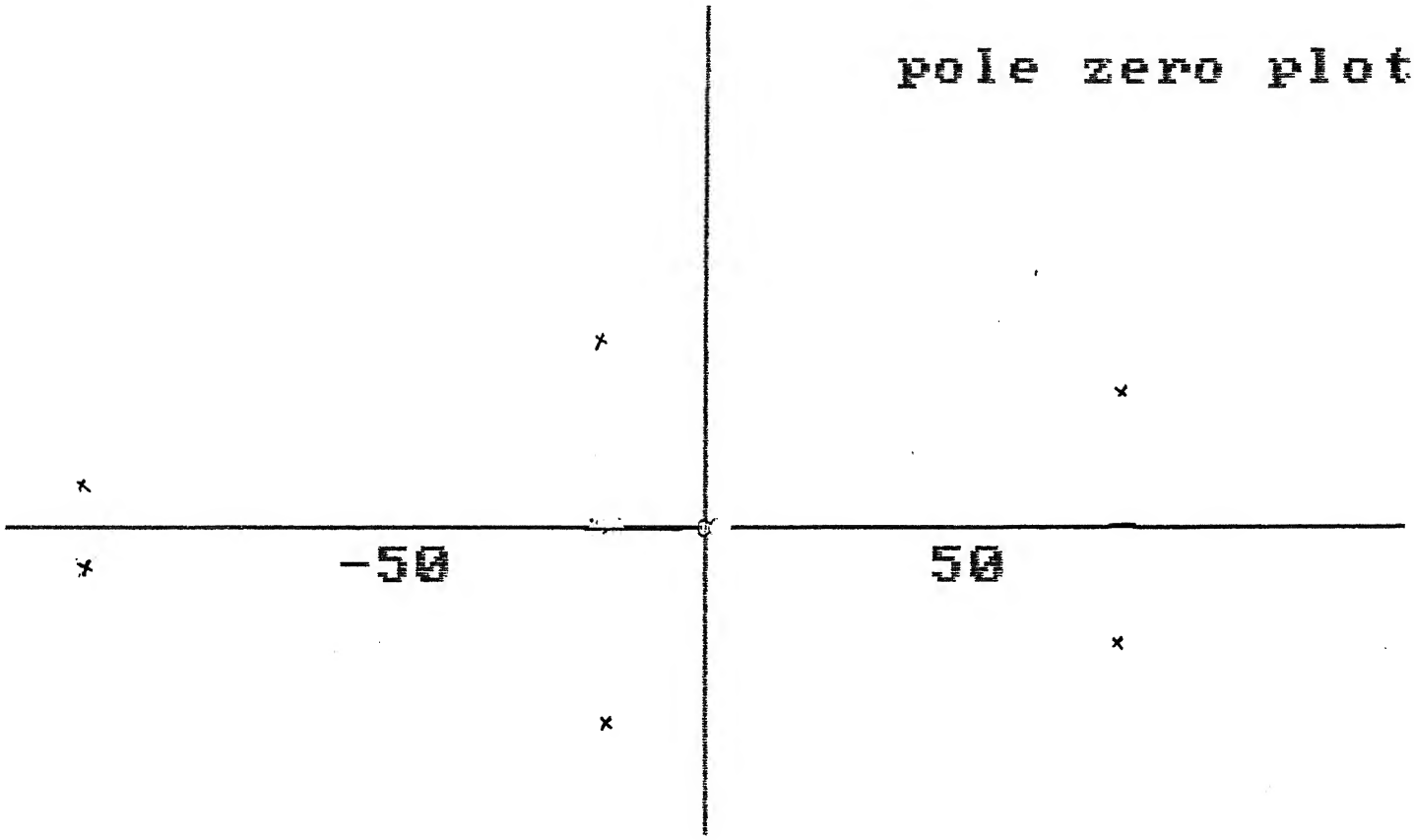


Figure 7.3.3

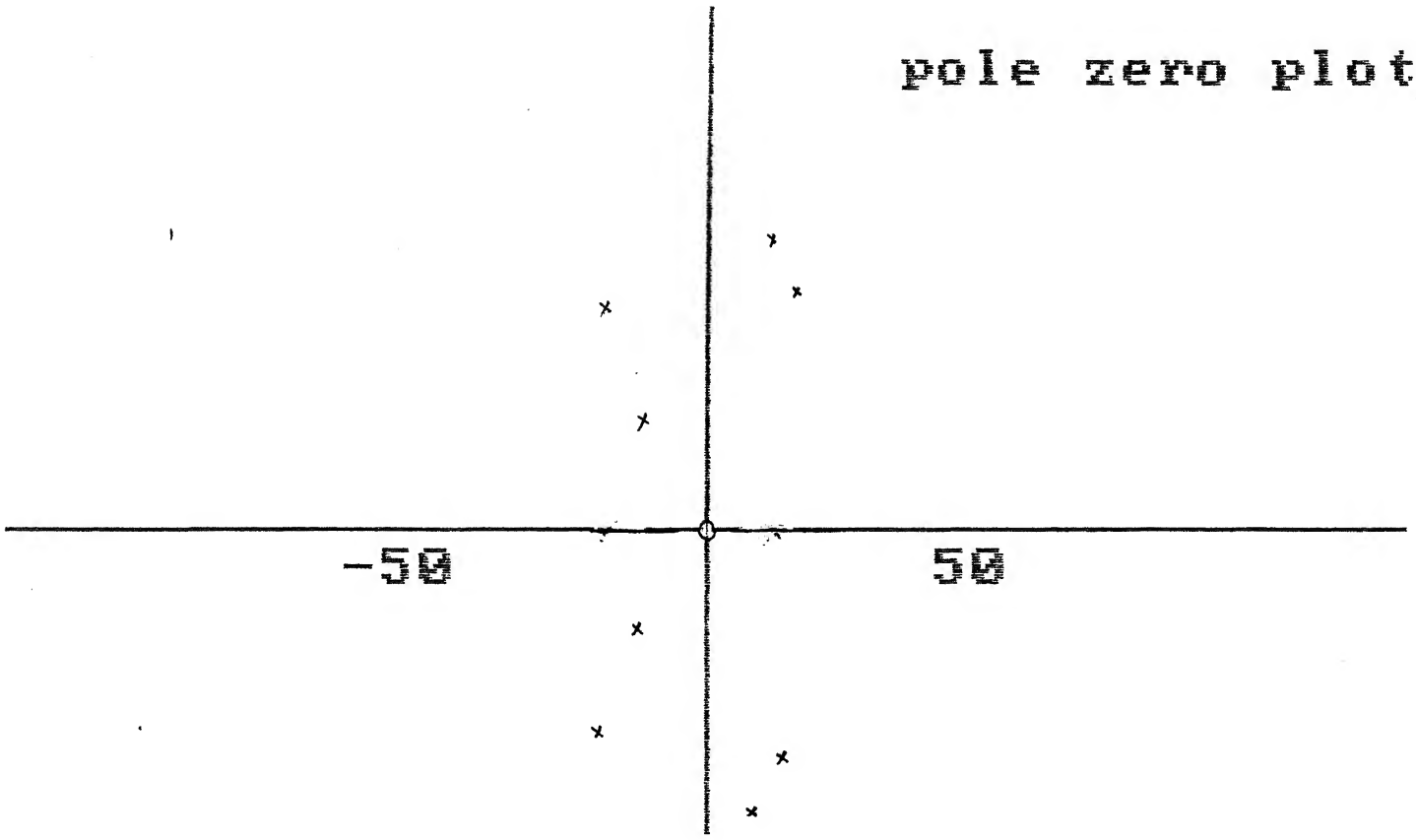


Figure 7.3.4

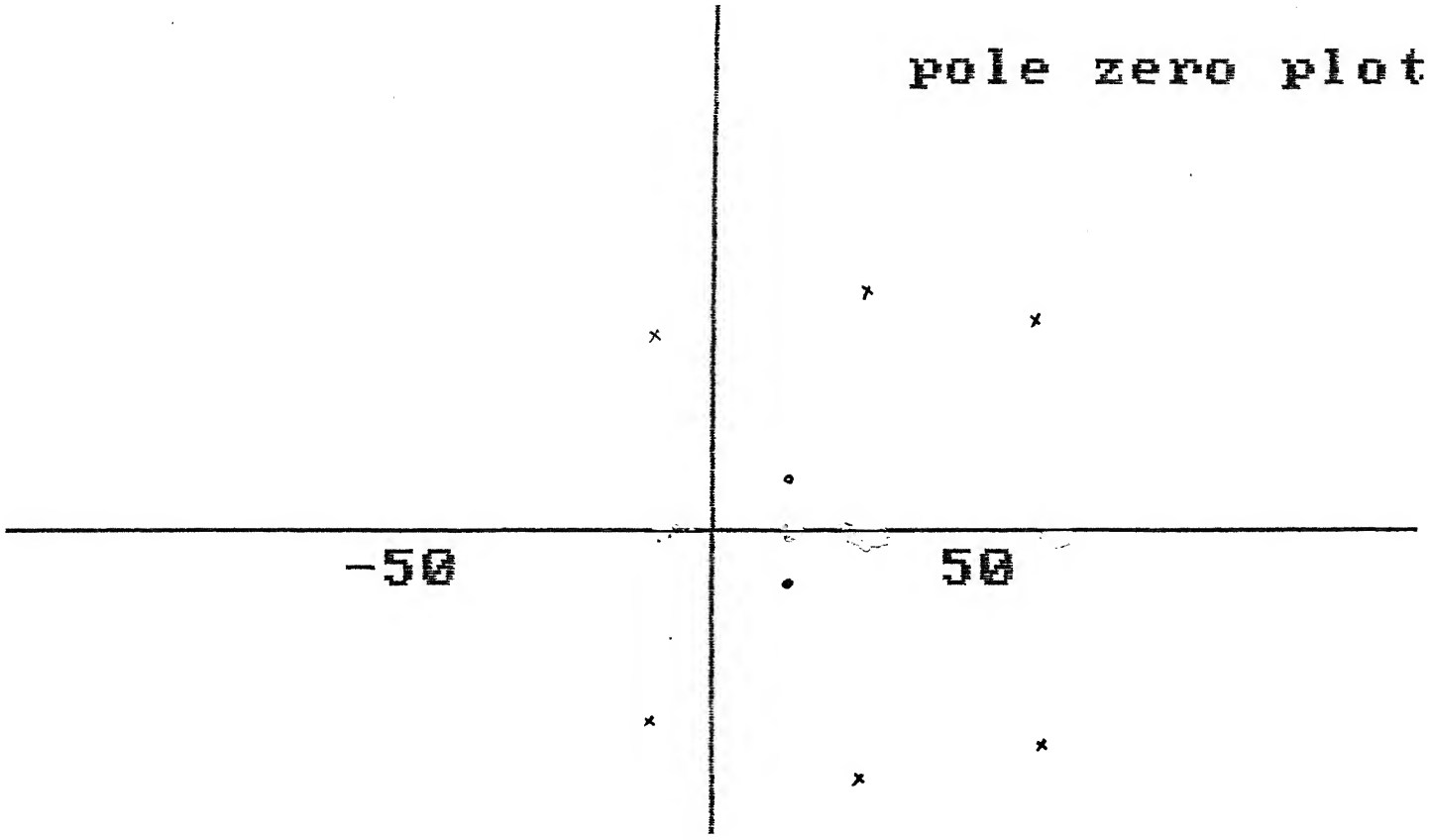


Figure 7.3.5

Waveforms	Zeros	Poles
1	0±j0	-0.887±j0.463; -0.21±j0.025 0.063±j0.55 ; 0.087±j0.56
2	0±j0	-0.38±j0.45 ; -0.21±j0.042 0.063±j0.57 ; 0.089±j0.55
3	0±j0	-0.42±j0.0344;-0.023±j0.064 0.463±j0.047;
4	0±j0	-0.42±j0.46; -0.373±j0.034 0.021±j0.56; 0.029±j0.55
5	0.034±j0.0466	-0.21±j0.42; -0.15±j0.5 0.16±j0.46

TABLE 7.1 (a)

List of Pole and Zero locations

(for plots 7.3.1 to 7.3.5)

1 st wave	2 nd wave	Distance
Vowels 1 to 4 Nasal 1	Vowel 1 to Vowel 4 Nasal 1	0 0
Vowel 1 Vowel 2 Vowel 3 Vowel 4 Nasal 1	Vowel 1 (CODEC O/P) Vowel 2 (do) Vowel 3 (do) Vowel 4 (do) Nasal 1 (do)	570 426 214 580 670
Vowel 1 Vowel 2 Vowel 3 Vowel 4 Nasal 1	Vowel 1 (noise with) (max value 50 added) Vowel 2 (do) Vowel 3 (do) Vowel 4 (do) Nasal 1 (do)	243 217 202 280 373
Vowel 1 Vowel 2 Vowel 3 Vowel 4 Nasal 1	Vowel 1 (noise with) (max value 100 added) Vowel 2 (do) Vowel 3 (do) Vowel 4 (do) Nasal 1 (do)	537 441 227 562 712
Vowel 1 Vowel 2 Vowel 3 Vowel 4 Nasal 1	Vowel 1 (expanded twice) Vowel 2 (do) Vowel 3 (do) Vowel 4 (do) Nasal 1 (do)	423 378 283 498 1283
Vowels 1 to 4 Nasal 1	Vowels 1 to 4 (expanded 3 times) Nasal (expanded 3 times)	croses range croses range

Table 7.1

Distance Comparisons of Two Waveforms by Pattern Matching

APPENDIX A

REGISTERS In ADSP 2100

REGISTER TYPE	DATA SIZE	FUNCTION
DATA REGISTERS		
AX0, AY0, AR, AX1, AY1, AF	all 16 bits	Used in ALU operations and as numerator and denominator in divide operations.
MX0, MX1	all 16 bits	Used in multiply accumulate... instructions.
MR0, MR1, MR2	MR2 4 bits... rest 16 bits	Stores results of multiply.... accumulate instructions.
SR0, SR1, SI, SE	all 16 bits	Used in shift instructions....
INDEX REGISTERS		
I0 to I3	all 16 bits	Used for address generation...
I4 to I7	all 16 bits	Used for address generation... but only these can fetch data from program memory data area
MODIFY REGISTERS		
M0 to M3	all 16 bits	Used as increment for post.... index modify instructions used only with I0 to I3
M4 to M7	all 16 bits	Used as above except that..... used only with I4 to I7

APPENDIX B

ADSP 2100 INSTRUCTION TYPES

Here the Classification is the Instruction groups and then the individual instructions possible within the group.

ALU

- Add / Add with Carry
- Subtract X-Y / Subtract X-Y with Borrow
- Subtract Y-X / Subtract Y-X with Borrow
- AND, OR, Exclusive OR
- Pass / Clear
- Negate, NOT
- Absolute value
- Increment, Decrement
- Divide

MAC

- Multiply
- Multiply / Accumulate
- Multiply / Subtract
- Clear
- Transfer MR

SHIFTER

Arithmetic Shift, Logical Shift

Normalize

Derive Exponent, Block Exponent Adjust

Arithmetic Shift Immediate

Logical Shift Immediate

MOVE

Register Move

Load Register Immediate

Data Memory Read (Direct Address)

Data Memory Read (Indirect Address)

Program Memory Read (Indirect Address)

Data Memory Write (Direct Address)

Data Memory Write (Indirect Address)

Program Memory Write (Indirect Address)

PROGRAM FLOW

JUMP, CALL

Return from Subroutine

Return from Interrupt

Do Until

TRAP

MISC

Stack Control

Mode Control

Modify Address Register

NOP

MULTIFUNCTION

ALU / MAC / SHIFT operation with Memory Read

ALU / MAC / SHIFT operation with Data Register Move

ALU / MAC / SHIFT operation with Memory Write

Data & Program Memory Read

ALU / MAC operation with Data & Program Memory Read

APPENDIX C

SYSTEM BUILDER DIRECTIVES

1) .SYSTEM system_name;

This directive is the first statement in the system specification file.

2) .ENDSYS;

This directive is the last statement of the system specification file. It terminates the System Builder processing at this directive.

3) .CONST <assignment>[,<assignment>,...];

This directives defines the System Builder constants. These constants can be used in place of numerical constants.

4) .PORT/ABS = address port_name;

It is used to declare memory mapped I/O port in the Data Memory.

5) .SEG/*loc*/ABS = address/type/[attribute,..] segment_name [constant];

This directive declares memory segment for program memory or data memory. The *loc* is PM or DM. The address is the absolute address of the segment.

The type is the type of memory RAM or ROM. The attribute is either CODE or DATA. The segment name is the identifer. It is a symbolic name. The constant

gives the size of the array.

APPENDIX D

ASSEMBER DIRECTIVES

1) .MODULE[[/ <module_qual>]..] <module name>;

It defines the start of an assembly module and is the first statement of the .DSP file. 'Module qual' is either RAM or ROM.

2) .ENDMOD;

It is the last assembler statement in the source code file. The assembly process is terminated after the ssembler reads this directive.

3) .VAR[[/ < var_qual >] < var_buffer > [,< var_buffer >]...

This directive is used to declare data buffers. 'Var_qual' specifies the
memory area PM or DM , RAM or ROM etc. 'Var_buffer' gives the buffer name and its length.

4) .CONST < assignment > [,< assignment >]...

This directive is used to declare symbolic constants.

5) .PORT < port_name >

It declares the memory mapped I/O port in the data memory.

6) .INIT < init_statement >

It is used to initialise a declared variable or all or part of a data buffer. 'Init_statement' gives the buffer name the file and the initialisation data.

7) .INCLUDE < filename >

t is used to read another source code file.

8) .MACRO < macro_name .> [(< macro_arg > [,]..)]

This directive is used to define a macro. 'Macro_name' gives the name of the macro.

9) .LOCAL < local_label >

It is used within a macro definition. It creates a unique label for the local
label at each invocation of the macro.

10) .EXTERNAL < external_symbol >

It assigns external attributes to identifers.

11) .GLOBAL < buffer_name >

It assigns global attributes to ports and variables.

12) .ENTRY < entry_label >

This directive assigns entry attributes to the label names.

APPENDIX E

SIMULATOR COMMANDS

Display Control Commands

ALternate	displays secondary data registers
BACkup	forces PM/DM/Trace displays to scroll back
BEep	enables beeps on user's terminal
CAche	invokes cache display mode
DECimal	forces all numbers to be displayed in decimal format
DM	invokes data memory display mode
FOrward	forces PM/DM/Trace displays to scroll forward
HELp	displays command list for access to help information
HEXadecimal	forces all numbers to be displayed in hexadecimal (the default)
Modules	displays all source modules
NOBeep	suppresses beeps at the user's terminal
NOSymbolic	forces the simulator to be non-symbolic
PLotdm	plots the contents of a selected section of DM
PM	invokes program memory display mode
PRimary	displays primary data registers
REGister	invokes register display mode
STACk	invokes stack display mode
STATus	displays Interrupt, Break, and Port status
SYmbolic	forces the simulator to be symbolic (default)
TOggle	toggles display of primary and secondary register banks
TRace	invokes trace display mode
Wipe	rewrites current display
Xreference	displays cross-reference list

Operation Control Commands

EMulator	invokes emulator mode
EXTend	invokes extend mode
SInglestep	invokes single-step mode

Break Control Commands

CLEARBreak	clears a PM break address
CLEARStoptime	clears any stop times currently defined
CLEARWatch	clears a DM access watch address
COunt	sets iteration count and delay on break points
SETBreak	sets a PM break address
SETStoptime	sets a time in ns for the processor to halt
SETWatch	sets a DM access watch address

Context Control Commands

SETModule	sets the module the Simulator uses for symbolic context

File Control Commands

COMmfile	executes simulator commands found in a batch file
DUMPDm	forces a DM image dump to a file
DUMPPm	forces a PM image dump to a file
Load	reads .EXE and .SYM file and sets default module context
READImage	reads a memory image file
READSymbol	reads a symbol table file

Modify/Inspect Control Commands

CLEARTime	clears the time display
CYCLetime	sets the cycle period in ns
FINDDm	finds the occurrence of a value in DM
FINDPm	finds the occurrence of a value in PM
RESEtstack	clears stacks and reset pointers
SETDm	sets a segment of DM
SETPC	sets the PC
SETPM	sets a segment of PM
SETRegister	sets a register value

Assembly Commands

ADdsymbol	adds a user-defined symbol name
DELete	deletes one line of assembly code from PM
EXEcute	executes an assembly instruction
PAtch	patches one line of assembly code into PM
REMovesymbol	deletes a user-defined symbol name

Configuration Control Commands

BATch	turns off screen update in Emulator mode
CHipreset	simulates the hardware chip RESET
CLOse	closes a DM memory mapped I/O port
HArdware	simulates hardware powerup and sets ROM to undefined
Interrupts	activates the interrupt source
Open	opens a DM memory mapped I/O port
POwerup	simulates the hardware powerup condition

Execution Control Commands

RUn	starts processor running in Extend and Emulator modes
<cr>	starts processor running in Single-step mode

Exit Command

EXIt	exits from the Simulator and returns to the host

APPENDIX F

EVALUATION BOARD COMMANDS

This section provides a summary of all Evaluation Board Commands.

System Control

ACCesscntr	sets the Evaluation Board to Accesscntr mode
CHipreset	asserts the RESET signal of the ADSP-2100A and also asserts the RESETOUT signal of the prototyping expansion connector (see Chapter 6)
FUllcntr	sets the Evaluation Board to Fullcntr mode
IBmpc	sets the Evaluation Board to IBM PC transmission mode
SETBAud	sets the baud rate for the RS-232C connections
SETBIts	sets the bits per character for the RS-232C connections
SETClock	sets the clock source for the Evaluation Board session
SETPArity	sets the parity status for the RS-232C connections
TRansparent	sets the Evaluation Board to transparent mode
^D	toggles between transparent mode and normal mode
Upload	transfers contents of Evaluation Board memory to host
Vax	sets the Evaluation Board to VAX transmission mode

Display Control

ALternate	displays secondary data registers
BAckup	scrolls PM/DM displays backward
BEep	enables beeps at the user's terminal
DECimal	inputs and displays all numbers in decimal
DM	invokes data memory display
FOrward	scrolls PM/DM displays forward
HELp	displays a list of all Evaluation Board commands
HEXadecimal	inputs and displays all numbers in hexadecimal
Modules	displays the names of all source modules in program memory
NOBeep	suppresses beeps at the user's terminal
NOSymbolic	forces the Evaluation Board to display the value of symbols
PM	invokes program memory display
PRimary	displays primary data registers
REGister	invokes register display
STACk	invokes stack display
STATus	displays break point status, audible tone mode, counter mode, transmission mode, and symbolic reference mode
SYmbolic	forces the Evaluation Board to display symbols, not their values
TOggle	toggles between primary and secondary data register displays
Wipe	rewrites current display
Xreference	displays a cross-reference list

Operation Control

EMulator	invokes Emulator mode
EXTend	invokes Extend mode
SInglestep	invokes Single-Step mode

Break Control

CLEARBreak	clears a program memory break point
SETBReak	sets a program memory break point

Context Control

SETModule	sets the module the Evaluation Board uses for symbolic context

Modify/Inspect Control

CLEARCycle	clears the cycle counter
FINDDm	finds the occurrence of a value in data memory
FINDPm	finds the occurrence of a value in program memory
RESetstack	clears stacks and resets pointers
SETDm	sets the value of a segment of data memory
SETPC	sets the value of the program counter
SETPM	sets the value of a segment of program memory
SETRegister	sets a register value

Assembly

DELete	deletes one line of assembly code from program memory
PAtch	patches one line of assembly code in program memory

Execution Control

<cr>	starts emulation in Single-Step mode
RUn	starts emulation in Extend and Emulator modes
EXEcute	executes an instruction (given in machine code)

REFERENCES

[1] *ADSP2100 Cross–Software Manual* of Analog Devices, 1988.

[2] *ADSP2100 Evaluation Board Manual* of Analog Devices, 1988.

[3] *ADSP2100 Application Handbook Volume 1* of Analog Devices, 1988.

[4] Rabiner, L. R and Schafer, R. W. "*Digital Processing of Speech Signals*", Prentice–Hall Inc., Englewood Cliffs, New Jersey, 1978.

[5] Oppenheim, A. V. "*Applications of Digital Signal Processing*", Prentice–Hall Inc., Englewood Cliffs, New Jersey, 1978.

[6] Flanagen, J. L. "*Speech Analysis, Synthesis and Perception*", Springer–Verlag, New York, 1972.

[7] Blakenship, P. E. "*A Review of Narrow Band Speech Processing Techniques*", Proceedings of the International Specialist Seminar on Case Studies in Advanced Signal Processing, Stockholm, September 1979, pp.108–118.

[8] Kleign, W. B., Krasinski, D. J. and Ketchum. R. H. "*Fast Methods for the CELP Speech Coding Algorithm*", IEEE Transactions on Accoustics Speech and Signal Processing, Vol. 38, No. 8, August 1990, pp.1330–1341.

[9] Markel, J. and Gray, A., "*Linear Prediction of Speech*", Springer–Verlag, New York, 1976.

[10] Makhoul, John "*Linear Prediction: A Tutorial Review*", Proceedings of IEEE, Vol. 63, No. 4, April 1975.

[11] Le Guyader, A. "*An efficient structure for speech coding between 8 kbits/s and 16 kbits/s*", Signal Processing IV: Theories and Applications, Lacoume. J. L.,*et al* Eds. Elsevier Science Publishers (North–Holland), EURASIP 1988, pp.867–870.

[12] Nandgopal, D. and Johnson, D. A. H. "*Pole–Zero Analysis of Speech by Linear Predictive Coding Using Minimum Phase Criteria*", ISSPA 87, signal processing, theories, Implementations and Applications., Brisbane Australia, August 1987, pp.618–622.

[13] Takizawa, Y., Tokuda, K. and Fukasawa, A. "*New Algorithm on Formant–Based Analysis and Pattern Matching of Voice*", Signal Processing IV: Theories and Applications, Lacoume. J. L. *et al* Eds., Elsevier Science Publishers B.V.(North–Holland), EURASIP, 1988, pp.535–538.

[14] Sastry, S. S. "*Introductory Methods of Numerical Analysis*", Prentice–Hall of India Pvt Ltd., 1990.

[15] Atal, B. S. "*Automatic Recognition of Speakers from their voices*", Procedings of IEEE, Vol. No 64, No. 4, April 1976, pp 460 to 475.

[16] Rosenberg, A. E. "*Automatic Speaker Verification:A Review*", Procedings of IEEE, Vol. No 64, No. 4, April 1976, pp 475

[17] Oppenheim, A. V. and Schafer, R. W. "*Digital Signal Processing*", Prentice–Hall of India Pvt Ltd.,1991.